Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikka
Tietojärjestelmien kehitys

Joonas Melanen

Rakennuksen tietomalli peliympäristönä

Opinnäytetyö 2014

Tiivistelmä

Joonas Melanen
Rakennuksen tietomalli peliympäristönä, 51 sivua, 2 liitettä
Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikka
Tietojärjestelmien kehitys
Opinnäytetyö 2014
Ohjaajat: tuntiopettaja Yrjö Utti, Saimaan ammattikorkeakoulu, lehtori Mikko Huhtanen, Saimaan ammattikorkeakoulu, lehtori Timo Lehtoviita, Saimaan ammattikorkeakoulu

Opinnäytetyössä tutkittiin ja testattiin rakennusten tietomallien hyödyntämistä pelien kehityksessä, tässä tapauksessa käytettiin valmiita tietomalleja yksinkertaisen hahmon silmistä kuvatun pelin ympäristönä. Opinnäytetyöhön kuului myös siinä käytettyjen ohjelmien opiskelu.

Opinnäytetyössä käytetyt tietomallit olivat IFC-tiedostomuodossa (Industry Foundation Classes). Nämä tietomallit muutettiin Unity-pelimoottorissa toimiviksi kolmiulotteisiksi ympäristöiksi käyttämällä Autodeskin 3ds Max Design 2012- sekä avoimen lähdekoodin Blender -3D-mallinnusohjelmia. Lopuksi itse pelit toteutettiin edellä mainitulla Unity-pelimoottorilla. Pelien ohjelmointiin käytettiin C#- ja Javascript-ohjelmointikieliä ja koodi kirjoitettiin Unityn mukana tulleella MonoDevelop-ohjelmalla.

Opinnäytetyön lopputuloksena on tämä raportti sekä kaksi Unitylla kehitettyä yksinkertaista peliä, joissa käyttäjä voi vapaasti tutkia kahdesta eri tietomallista rakennettuja peliympäristöjä. Raportti on tarkoitettu sekä pelialasta kiinnostuneille että tietomallien kehittäjille ja suunnittelijoille, jotka haluavat lisää tietoa kehittelemiensä mallien käyttömahdollisuuksista.

Asiasanat: Unity, 3D-mallinnus, tietomalli, pelinkehitys

Abstract

Joonas Melanen
A Building Information Model as a game environment, 51 Pages, 2 Appendices
Saimaa University of Applied Sciences
Technology Lappeenranta
Information Technology
Information System Development
Bachelor´s Thesis 2014
Instructor(s): Lecturer Yrjö Utti, Senior Lecturer Mikko Huhtanen, Senior Lecturer Timo Lehtoviita

In this thesis, the utilisation of Building Information Models (BIMs) in games development was examined and tested. Previously made BIMs were used as environments for a simple game controlled from a first-person viewpoint. Learning to use the programs used to develop the game was also a part of the thesis.

The information models used in this thesis were in the IFC (Industy Foundation Classes) file type. These information models were changed into three dimensional (3D) environments using Autodesk's 3ds Max Design 2012 and the open software 3D-modelling software Blender. The games themselves were made with the Unity game engine. The programming languages C# and Javascript were used to program the games; this code was written with MonoDevelop, a program that came with Unity.

The end result of this thesis is this report and two games developed with Unity, in which the player can freely explore their surroundings built from two different information models. This thesis is meant for those interested of games development and for the developers and designers of information models who want to know more about the other possible uses of their models.

Keywords: Unity, 3D-modelling, information model, game development

Sisältö

Termit ja käsitteet

.blend	Blenderin vakio tiedostomuoto.
.fbx	FilmBoX, Autodeskin omistama tiedostomuoto.
3D-malli	”Three Dimensional”, kolmiulotteinen esitys esineestä, hahmosta tai ympäristöstä näytöllä.
3D-Mallinnus	Kolmiulotteisten mallien luominen esimerkiksi rakennuksista.
Akseli	Kuviteltu suora, jonka ympäri kappale pyörähtää tai jonka suhteen kappaleen osat sijaitsevat symmetrisesti tai muutoin tietyllä tavalla.
Aluevalo (Area Light).	Valoa, joka lähtee neliön muotoisen tason toiselta puolelta tasaisesti. Nämä valot ovat liian raskaita järjestelmälle, joten niitä ei käytetä reaaliaikaisesti.
Animointi	Liikkuvan kuvan luominen malleja käyttämällä.
Android (käyttöjärjestelmä)	Googlen omistama käyttöjärjestelmä. Pohjautuu avoimeen lähdekoodiin.
App Store	Applen tuotteiden oma kauppapaikka, josta asiakas voi ostaa omille Applen laitteilleen erinäisiä sovelluksia (Appeja).
Apple	Yhdysvaltalainen suuryritys, joka suunnittelee, kehittää ja myy kulutuselektroniikkaa, ohjelmistoja ja tietokoneita. Sen tunnetuimpia tuotteita ovat muun muassa Macintosh-tietokoneet sekä lukuiset i-laitteet (iPhone, iPad, iPod...).
ArchiCAD	Rakennussuunnittelijoille tarkoitettu mallinnustyökalu. Sitä voidaan käyttää kolmiulotteisten rakennusmallien luontiin. Opinnäytetyössä käytetyistä tietomalleista toinen tehtiin tällä ohjelmalla.
Asset	Unityssa pelin kehittämiseen käytettävä elementti. Voi olla esimerkiksi 3D-malli, tekstuuri, musiikkinäyte tai vastaava.
Autodesk	Erinäisiin mallintamisohjelmiin erikoistunut ohjelmistoyritys. Autodeskin tuotteisiin kuuluvat muun

	muassa AutoCAD, ArchiCAD, 3ds Max ja 3ds Max Design.
Blender	Avoimen lähdekoodin 3D-mallinnusohjelmaa
C#	Microsoftin kehittämä oliopohjainen ohjelmointikieli.
Character Controller	Termi Unityssa, pelihahmolle tietyt parametrit antava ominaisuus.
Collider	Törmäytin, tekee objektista kiinteän Unityssa (pelaaja ei voi liikkua objektin läpi).
Field of View	Näkökenttä. Kulma, jonka verran (tässä tapauksessa) pelaajan hahmo näkee eteensä tai ympärilleen.
Fysiikan mallinnus	Pelin sisäisen fysiikan (esimerkiksi painovoiman, objektien massan vaikutuksen niiden liikkumiseen, tavaroiden putoamisnopeuden) mallintaminen peliin. Useissa pelimoottoreissa tämä on valmiiksi tehty.
GameObject	Termi Unityssa, pelin sisäinen objekti.
GNU-lisenssi	Vapaiden ohjelmistojen julkaisemiseen tarkoitettu lisenssi. Antaa kenelle tahansa oikeuden käyttää, kopioida, jakaa ja muuttaa ohjelmia ja niiden lähdekoodia.
Hierarkia	Kuvaa pelin kentän sisältämät objektit ja niiden suhteet toisiinsa.
Indie-kehittäjä	”Independent”, itsenäinen kehittäjä, pelinkehitystä ilman suurta taloudellista tukea videopelien julkaisijalta. Kehitykseen kuluneessa ajassa indiepelit voivat vaihdella vuosista tunteihin.
Infranhaltija	Yritys tai taho, joka omistaa jonkin kohteen infrastruktuurin.
Infratietomalli	Tietomalli, joka kuvaa infrastruktuurin toimintaa.
Inspector	Unityssa oleva käyttöliittymän osa, joka listaa tietoa valitusta objektista.
IOS	Applen kehittämä käyttöjärjestelmä.
Iphone	Applen kehittämä älypuhelinmalli.
Jälkikäsittely	Jälkikäsittelyn seurauksena 3D-malli saa lopullisen muotonsa.
Kohdevalo (Spot Light).	Kartion muotoinen valo, joka loistaa yhteen suuntaan. Käytetään esimerkiksi taskulamppuja tehdessä.

Konsoli	Näyttää peliä suoritettaessa ilmentyvät ongelmat ja varoitukset pelin koodista.
Kontrollit	Pelihahmon liikuttamiseen ja ohjaamiseen tarkoitettujen toimintojen ohjeet.
Käyttöliittymä	Ohjelman hallintaan ja käyttöön tarkoitettu osa ohjelmaa.
Lapsi	Objekti jolla on vanhempi tai vanhemmat. Lasta voi muuttaa vaikuttamatta muihin lapsiin tai vanhempaan.
Liipaisin (trigger)	Objekti, jolle voidaan asettaa komento joka tapahtuu sen reagoidessa muiden objektien kanssa.
Linux	GNU-projektin alunperin kehittämä avoimen lähdekoodin käyttöjärjestelmä, jota kaikki käyttäjät ovat vapaita muokkaamaan.
Lähetin	Laite, joka lähettää kuvaa radiosignaalin välityksellä. Yhteydessä vastaanottimeen.
Mac	Applen omistamat Macintosh-tietokoneet ja laitteet
Mesh Renderer	Luo objektin ulkoasun kentässä.
MonoDevelop 4.01	Unityn mukana toimitettu, alustasta riippumaton ohjelmointiympäristö.
Muuttuja	Ohjelmoinnissa jonkin arvon varastoiva yksikkö.
NaN	Ton Roosendaalin vuonna 1998 perustama yritys. Nimi tulee sanoista Not a Number. Yrityksen tarkoituksena oli kehittää ja markkinoida Blenderiä.
Oliopohjaisuus	Ohjelmoinnin lähestymistapa, jossa ongelmien ratkaisut jäsennetään olioiden (ohjelmoinnin perusyksikkö, joka sisältää yhteenkuuluvaa tietoa ja toiminnallisuutta) avulla.
Paint	Microsoftin kuvankäsittelyohjelma.
Pause	Tauko. Käytetään testattavana olevan pelin tauottamiseen.
Pelihahmo	Pelin hahmo, jota pelaaja ohjaa.
Pelikonsoli	Pääasiassa videopelien pelaamiseen tarkoitettu laite.
Pelitalo	Pelien kehittämiseen erikoistunut yritys.

Pikanäppäin-komento	Yhden tietyn toiminnon suorittava näppäin tai näppäin-yhdistelmä (esimerkiksi Control + S tallentaa tiedoston).
Pistevalo (Point Light).	Pisteen muotoinen valo, joka loistaa valoa tasaisesti kaikkiin suuntiin.
Play	Pelaa tai toista. Käytetään kehitettävän pelin testaamiseen.
Position	Sijainti. Käytetään muuttamaan objektin sijaintia.
Reititin	Tietoverkkoja yhdistävä laite.
Renderöinti	Kuvan luomista mallista tietokoneohjelman avulla.
Rotation	Kääntö. Käytetään kääntämään objektia akselin ympäri.
Scale	Skaala. Käytetään muuttamaan objektin skaalaa eli kokoa.
Scene eli kenttä	Pelin kenttä. Sisältää kentässä käytettävät objektit.
Skripti	Komentokehoite, käytetään ohjaamaan pelin toimintaa.
Smartboard	Kosketustaulu. Tauluun voidaan kiinnittää tietokone, jolloin tietokonetta voi käyttää kosketustaululla.
Sovellus	Ohjelma, joka on tarkoitettu käyttäjälle.
Step	Askel. Käytetään testattavana olevan pelin edistämiseen ruutu kerrallaan.
Suuntavalo (Directional Light).	Suuntavalo loistaa yhteen suuntaan tasaisesti kaikkialle kenttään. Sitä käytetään pääasiassa simuloimaan auringon tai vastaavan tuottamaa valoa.
Tekstuuri	Kaksiulotteinen kuva, jota käytetään kolmiulotteisen mallin kuvioimiseen
Totuusarvo (boolean)	Arvo, joka voi olla joko tosi tai epätosi.
Vanhempi	Objekti jolla on lapsia. Muutokset vanhempaan vaikuttavat kaikkiin sen lapsiin.
Vastaanotin	Laite, joka ottaa lähetetyn signaalin vastaan. Yhteydessä lähettimeen.
Verkkoselain	Internetin selaamiseen tarkoitettu ohjelma.

Windows	Microsoftin käyttöjärjestelmä.
Xbim Xplorer	Avoimen lähdekoodin IFC-tiedostojeen tarkasteluun tarkoitettu tiedosto. Siitä on sekä itsenäinen että verkossa toimiva versio.
Zoomaus	Tiivistämistä edessä olevaan kohteeseen.

1 Johdanto

Tässä luvussa käydään läpi opinnäytetyön tausta, sen tavoite ja käydään työn rakenne läpi ennalta.

1.1 Työn tausta

Opinnäytetyön aiheena on tutkia ja testata rakennuksen tietomallien hyödyntämistä pelin kehityksessä. Tässä tapauksessa valmiita tietomalleja käytettäisiin yksinkertaisten, pelihahmon silmistä kuvattujen pelien ympäristöinä. Opinnäytetyössä tutkitaan myös, mitä lisäominaisuuksia Unity tuo tietomallin esittelyyn.

Peliala on nyt Suomessa nousussa ja kiinnostus alaa kohtaan vain kasvaa. Lisäksi pelien kehitys ja rakentaminen on nykyisin helpompaa aloittaa tarjolla olevien eri pelimoottorien ansiosta.

Pelimoottori on ohjelma, joka sisältää valmiiksi monia pelien ja niiden kehityksen vaatimia ominaisuuksia, kuten fysiikan mallinnus ja grafiikan piirtäminen. Pelitalot voivat itse valmistaa pelimoottorinsa tai ostaa valmiin sellaisen. Itse pelin sisältö, ulkonäkö ja pelissä käytettyjen osien reagointi toisiinsa muodostavat pelin. Nämä on pelin kehittäjän valmistettava itse (Ward 2008, 1).

Tässä opinnäytetyössä käytetty Unity on yksi näistä pelimoottoreista. Unityssa käytettävä sisältö saatiin koululta saaduista ja sen tiloja kuvaavista rakennuksen tietomalleista.

Opinnäytetyö liittyy Saimaan ammattikorkeakoulun tietomallintamiseen liittyvään kehityshankkeeseen. Tätä hanketta edustaa opinnäytetyössä lehtori Timo Lehtoviita.

1.2 Työn tavoite

Työn tavoitteena on kehittää kaksi yksinkertaista, hahmon silmistä kuvattua tietokonepeliä Unity-pelimoottorilla. Pelien ympäristöinä käytetään koululta saatuja tietomalleja, jotka muutetaan Unityn tunnistamiksi kolmiulotteisiksi

malleiksi eli 3D-malleiksi (Three Dimensional) käyttäen kahta eri 3D-mallinnusohjelmaa: Autodeskin 3ds Max Design 2012:ta ja avoimen lähdekoodin Blender-ohjelmaa. Pelien valmistusvaiheet esitellään vaiheittain. Työ on tarkoitettu pelien kehityksestä kiinnostuneille sekä tietomallien suunnittelijoille tai kehittäjille, jotka haluavat lisää tietoa siitä, miten heidän kehittelemiään malleja voidaan hyödyntää.

1.3 Työn rakenne

Toinen luku kertoo työssä käytetystä Unity -pelimoottorista. Luvussa perustellaan Unityn valinta opinnäytetyön toteutuksessa käytetyksi pelimoottoriksi ja kerrotaan hiukan teoriaa ja taustaa itse Unity3D-pelimoottorista.

Kolmas luku kertoo tietomalleista. Luvussa käydään läpi tietomallin idea, kerrotaan rakennusalan oliopohjaisen tiedon siirron standardista IFC (Industy Foundation Classes) sekä kerrotaan lyhyesti opinnäytetyössä IFC-mallien tarkasteluun käytetyistä ohjelmista.

Neljännessä luvussa kerrotaan työssä käytetyistä kahdesta 3D-mallinnusohjelmasta, Autodeskin 3ds Max Design 2012:sta ja Blenderistä. Kappaleessa kerrotaan myös, mitä ohjelmilta vaadittiin, jotta niissä voisi käyttää IFC-tietomalleja.

Viidennessä luvussa kerrotaan itse pelien valmistuksesta. Luvussa käydään läpi 3D-mallien siirtäminen Unityyn, niiden käyttäminen Unityssa, niiden ulkoasuihin ja esitykseen vaikuttavat muutokset ja tekijät, liikutettavan hahmon lisääminen peliympäristöön, hahmon liikuttamiseen ja ympäristön kanssa reagoimiseen tarvittavat skriptit ja lopuksi tuotoksen julkaiseminen.

Kuudennessa luvussa kerrotaan opinnäytetyön lopputuloksista ja pohditaan mahdollisia jatkokehitysmahdollisuuksia.

2 Unity-pelimoottori

Tässä luvussa esitellään Unity-pelimoottorin taustaa, sen käyttöliittymä ja sen ominaisuuksia sekä selitetään, miksi Unity valittiin tätä opinnäytetyötä varten ja mitä se tuo tietomallien esittelyyn.

2.1 Unityn taustatietoja

Unityn kehitys aloitettiin 2000-luvun alkupuolella kolmen ohjelmoijan toimesta: David Helgason, Joachim Ante ja Nicholas Francis. Kolmikko työskenteli Tanskassa. He lähtivät tekemään Unitya ideanaan tehdä jotain Applen Final Cut Pro:n kaltaista (Final Cut Pro antaa aloitteleville elokuvantekijöille järkevästi hinnoiteltuja, ammattimaisia elokuvien tekemiseen tarkoitettuja työkaluja) pelien kehittelijöille. (Brodkin 2013.)

Ensimmäinen versio Unitysta julkaistiin vuonna 2005. Kehitys jatkui ja 2008 pelimoottori oli hienostuneempi ja ohjelmien myynnistä saadut tulot maksoivat kehityskustannukset takaisin. Käännekohta tuli vuoden 2008 keskikohdilla, kun Apple julkaisi iPhonen App Store -kauppapaikan. Unityyn lisättiin nopeasti tuki iPhonelle, ollen näin ensimmäinen pelimoottori, joka saavutti sen. Tämän ansiosta Unityn suosio kasvoi huomattavasti. (Brodkin 2013.)

Siitä eteenpäin Unity on kasvanut huomattavasti, ja nykyään se työllistää 285 henkilöä globaalisti. Unity tukee kehitystä iOS:lle (Applen käyttöjärjestelmä), Android-käyttöjärjestelmälle, Windowsille, Macille, Linuxille, verkkoselaimille ja lukuisille konsoleille. (Brodkin 2013.)

2.2 Unity tässä opinnäytetyössä

Unity valittiin tähän opinnäytetyöhön sen yksinkertaisuuden, hinnan (Unitysta on saatavilla ilmainen, virallinen versio, joka on riisutumpi malli Unityn täydestä versiosta, Unity Pro:sta) ja sen tukemien ohjelmointikielien (C# ja Javascript) takia (Unity Technologies 2013).

Unityssa luotu peli saadaan toimimaan monilla eri alustoilla. Opinnäytetyössä Unityssa luodut pelit testattiin Windows-sovelluksina. Tämä tarkoittaa, ettei Unityssa luotu peli tarvitse ylimääräisiä ohjelmia toimiakseen: kunhan pelin

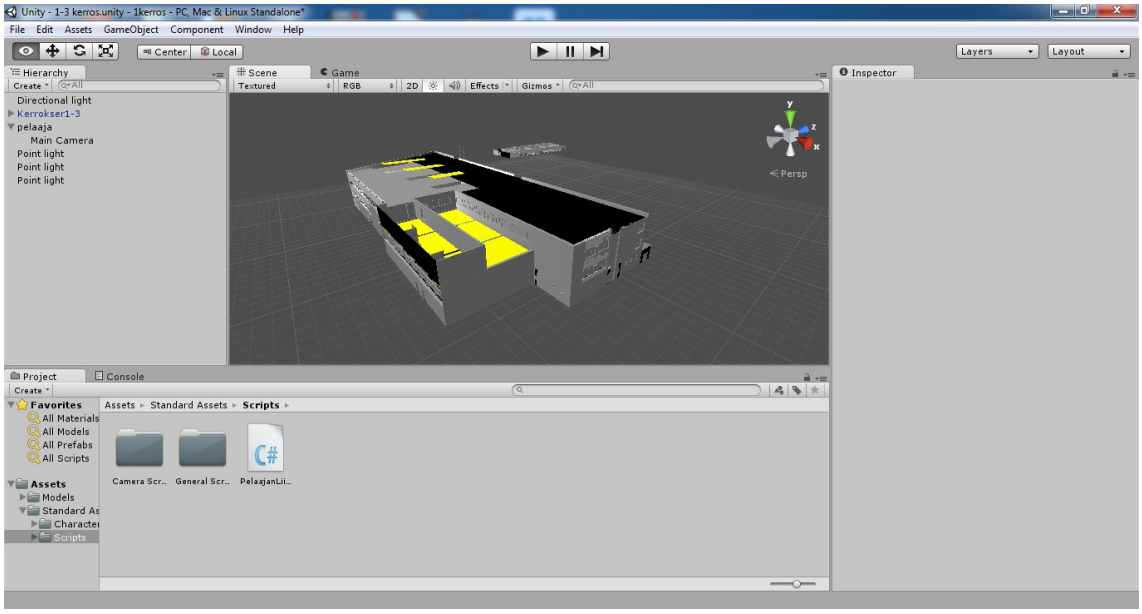
käynnistävä pikakuvake ja sen materiaalit sisältävä kansio ovat samassa kansiossa, peli toimii.

Tämän ansiosta rakennuksen tietomallia voidaan esitellä helposti ja käytännöllisemmin kuin jos lähdettäisiin esittelemään pelkkää tietomallia jossain sitä tukevassa ohjelmassa. Nämä ohjelmat eivät aina tue hahmon silmistä kuvattua liikkumista (niitä ei ole tarkoitettu sitä varten) ja tällöin tilan esittely jää hiirellä käänneltävän ja pyöriteltävän mallin esittelyksi. Tämä voi olla asiakkaan kannalta epäkäytännöllistä, jos yritetään esitellä tilaa, jota ihmiset käyttävät käyttäjän näkökulmasta.

Lisäksi tietomalli vaatii yleensä sen esittelyyn tarkoitetun ohjelman. Unityssa luotu Windows-peli testattiin tavallisella toimistokoneella, eikä tässä esiintynyt laitekohtaisia ongelmia eikä peli vaatinut ylimääräisten ohjelmien asentamista koneelle. Tämän ansiosta pelin jakaminen olisi huomattavasti helpompaa kuin tietomallin.

2.3 Unityn käyttäminen

Unityn käyttö pohjautuu sen graafiseen käyttöliittymään. Pelin käyttöliittymää, objekteja, elementtejä ja osia hallitaan hiirellä tai vaihtoehtoisilla pikanäppäin-komennoilla. Kuvassa 1 on esimerkki Unityn käyttöliittymästä.



Kuva 1. Unityn käyttöliittymä.

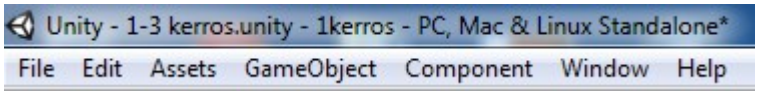
Unityn käyttöliittymä koostuu peruskäytössä seitsemästä osasta:

- valikkopalkista
- erinäisistä toimintopainikkeista
- pelin testaamiseen tarkoitetuista painikkeista
- pelin kentän hierarkiasta
- pelin näkymästä
- alalaidan suorakulmiosta, jossa voidaan näyttää erinäisiä pelin sisältöön liittyviä toimintoja, näkymiä tai valikoita
- valitun kohteen tutkijasta, ”Inspectorista”.

Nämä eri osat käydään yksitellen läpi seuraavissa aliluvuissa.

2.3.1 Valikkopalkki

Valikkopalkki sisältää kaikki Unityn käyttöön tarkoitetut ominaisuudet ja valikot. Siitä voidaan hallita muokattavia kenttiä ja projekteja ja luoda uusia objekteja, skriptejä ja vastaavia peliin. Lisäksi siitä voidaan lisätä peliin erinäisiä ominaisuuksia, avata uusia pelin hallintaan liittyviä ikkunoita ja lukuisia muita toimintoja. Valikkopalkki näytetään kuvassa 2.



Kuva 2. Unityn valikkopalkki.

Unityn valikkopalkki on välttämätön osa Unityn käyttöä.

2.3.2 Toimintopainikkeet

Toimintopainikkeita käytetään Unityssa sillä hetkellä muokattavana olevan kentän ja sen objektien liikuttamiseen ja manipuloimiseen sekä tarkasteluun. Vasemmanpuoleisin painike on tarkoitettu kentän näkymän hallitsemiseen. Käsi liikuttaa koko näkymää ja silmä pyörittää näkymän tarkasteluun käytettävää

kameraa. Nämä toiminnot saadaan myös tehtyä käyttämällä hiiren rulla-näppäintä ja hiiren oikeaa näppäintä.

Toinen painike vasemmalta on tarkoitettu pelin objektien liikuttamiseen. Kun se on valittu, valitun objektin ympärille ilmestyy kolme nuolta kuvaamaan kolmea eri akselia, joiden mukaan objektia liikutetaan: sininen nuoli on x-akseli, vihreä nuoli y-akseli ja punainen nuoli z-akseli.

Seuraava painike (keskimmäinen oikeanpuoleisista, kolmas vasemmalta) on tarkoitettu pelin objektin kääntämiseen eri akselien mukaan.

Oikeanpuolisin painike on tarkoitettu objektin skaalan muuttamiseen. Tässäkin objektin skaalaa voidaan muokata akseleittain.

Toimintapainikkeet näytetään kuvassa 3.

Kuva 3. Unityn toimintapainikkeet.

Kaikkia näitä kolmea objektin muokkaamiseen tarkoitettua toimintoa voidaan hallita myös valitun objektin tutkimiseen tarkoitetusta ”Inspectorista” antamalla sen Position-, Rotation- ja Scale-kohtien eri akseleita vastaaviin tekstikenttiin halutut arvot.

2.3.3 Pelin testaamiseen tarkoitetut painikkeet

Pelin testaamiseen on kolme painiketta Unityn käyttöliittymän ylälaidassa: ”Play” eli pelaa tai toista, ”Pause” eli tauko ja ”Step” eli askel.

Play käynnistää sen hetkisen kentän testauksen. Kun pelin testaus on käynnissä, painamalla Playta uudestaan pelin testaus voidaan lopettaa.

Pause tauottaa pelin juuri siihen hetkeen, joka siinä sillä hetkellä on tapahtumassa.

Step edistää pelin tapahtumia yhden ruudun kerrallaan. Sitä voidaan käyttää esimerkiksi efektien ulkoasun tarkistamiseen.

Kuvassa 4 näytetään nämä kolme painiketta.

Kuva 4. Pelin testaamiseen tarkoitetut painikkeet.

Näistä kolmesta painikkeesta Pausea ja Steppiä ei käytetty opinnäytetyössä ollenkaan; kaikki testaaminen suoritettiin Play -painikkeen avulla.

2.3.4 Pelin kentän hierarkia

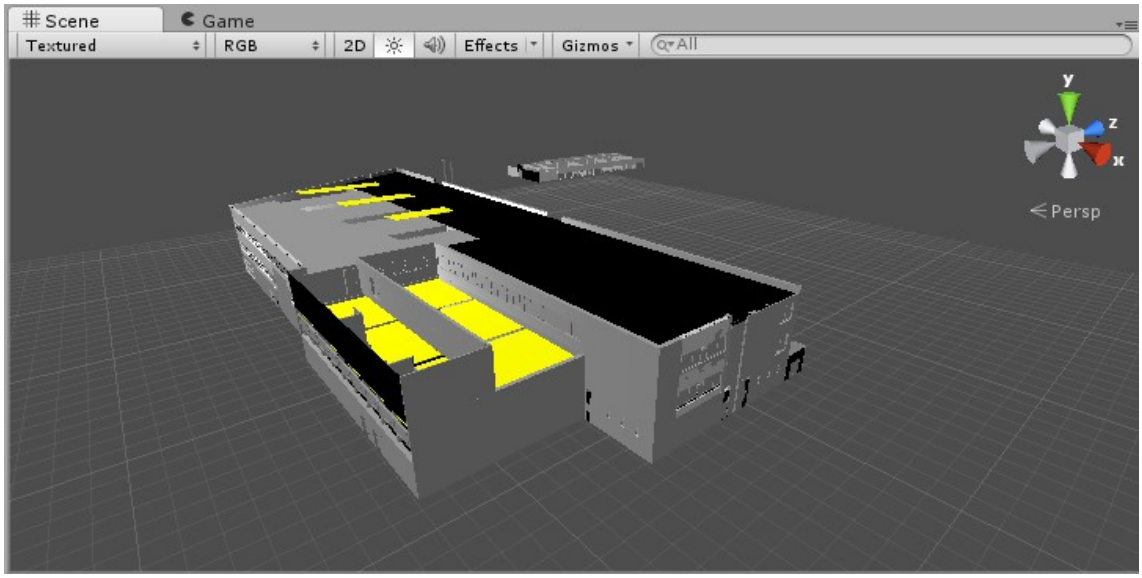
Hierarkia kuvaa pelin kentän sisältämät objektit ja niiden suhteet toisiinsa. Objektit voidaan asettaa toistensa ”lapsiksi”, jolloin muutokset ”vanhempaan” vaikuttavat samalla lapsiin. Kuvassa 5 koko pelin pääkamera ”Main Camera” on ”Pelaaja”-objektin lapsena.

Kuva 5. Pelin kentän hierarkia.

Koska kyseessä on hahmon silmistä kuvattu peli, täytyy kameran liikkua pelaajan mukana.

2.3.5 Pelin näkymä

Pelin kentän näkymä näyttää pelin kentässä olevat objektit. Se toimii samalla pelin kehittäjän näkökulmana peliin, pelissä itsessään olevan kameran kautta. Pelin kentän näkymä näytetään kuvassa 6.



Kuva 6. Pelin näkymä.

Painamalla ”Scene”- tai ”Game”-painikkeita pelin kehittäjä voi vaihtaa näkymää joko pelin kentän tai pelissä itsessään olevan kameran välillä. Peliin voidaan lisätä uusia objekteja raahaamalla niitä näkymään.

2.3.6 Alalaidan suorakulmio

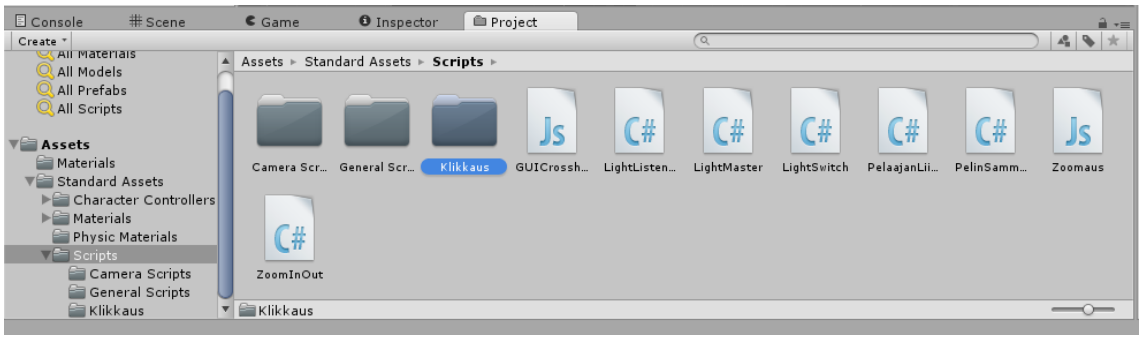
Alalaidan suorakulmion sisältöä voidaan vaihtaa painamalla sen ylävasemmassa laidassa olevia painikkeita. Nämä painikkeet ja niiden sisällöt ovat:

- ”Console” eli konsoli. Tämä näyttää peliä suoritettaessa ilmentyvät ongelmat ja varoitukset pelin koodista.
- ”Scene” eli kenttä tai kohtaus. Tämä kuvaa pelin sen hetkistä kenttää. Se on näkymän oletus.
- ”Game” eli peli. Tämä näyttää pelin näkymän pelin sisäisestä kamerasta.

- ”Inspector” eli tutkija. Tämä näyttää pelin sillä hetkellä muokattavana olevan objektin tiedot.

- ”Project” eli projekti. Tämä näyttää muokattavana olevan projektin kansioiden sisällöt. Tätä voidaan käyttää lisäämään tai siirtämään joku objekti tai skripti haluttuun kansioon projektin sisällä.

Tämä kohta näytetään kuvassa 7.

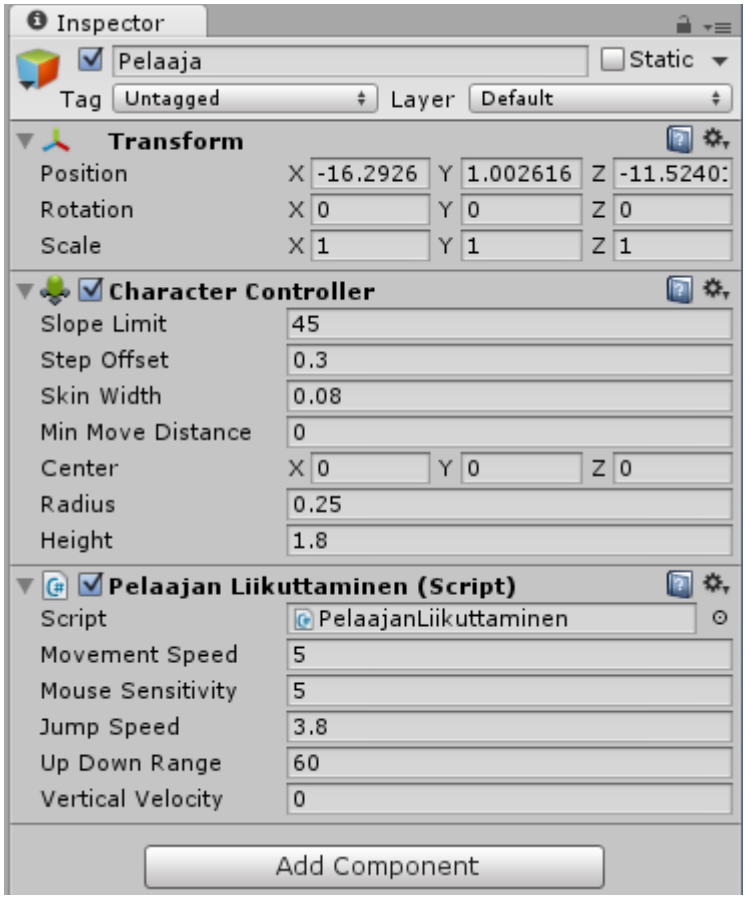


Kuva 7. Unityn käyttöliittymän alalaidan suorakulmio.

Tätä suorakulmiota voidaan siis käyttää erillisenä vaihtoehtona näyttämään muita projektin toiminnallisuuteen vaikuttavia valikoita tai toimintoja.

2.3.7 Inspector eli tutkija

”Inspector” eli tutkija näyttää sillä hetkellä valitun objektin tiedot, objektiin liitetyt komponentit (kuten hahmon liikuttaja, Character Controller) ja objektiin liitetyt skriptit. Tämä kohta näytetään kuvassa 8.



Kuva 8. Pelaaja-objektin Inspector eli tutkija.

Kuvassa ”Pelaaja”-objektiin on liitetty ”Character Controller”-komponentti ja ”PelaajanLiikuttaminen”-skripti, joka mahdollistaa pelihahmon liikuttamisen pelissä. Inspectorissa olevia arvoja muuttamalla voidaan vaikuttaa suoraan kohteena olevaan objektiin.

3 Tietomalli

Tässä luvussa kerrotaan tietomallin idea ja esitellään IFC -tiedostotyyppi ja sen tarkasteluun ja luontiin opinnäytetyössä käytettyjä ohjelmia.

3.1 Tietomallin kuvaus

Tietomalli on kohteena olevan tuotteen (tässä opinnäytetyössä koulun tilojen) ja rakennusprosessin koko elinkaaren aikaisten tietojen yhdistetty kokonaisuus digitaalisessa muodossa. Tietomallilla voidaan havainnollistaa suunnitelmat todenmukaisten 3D-mallien avulla. Lisäksi suunnittelun osa-alueet voidaan sovittaa yhteen paremmin ja tietomalliin voidaan sisältää enemmän tietoa, joka kaikki säilötään yhdessä tiedostossa. (Savisalo 2012, 2-4.)

Rakennussuunnittelussa tietomallinnus on jo vakiintunut työskentelytapa. Siinä käytettynä tiedostoformaattina on pääasiassa IFC ja ohjelmina esimerkiksi Autodeskin Revit ja ArchiCAD (Savisalo 2012, 7.).

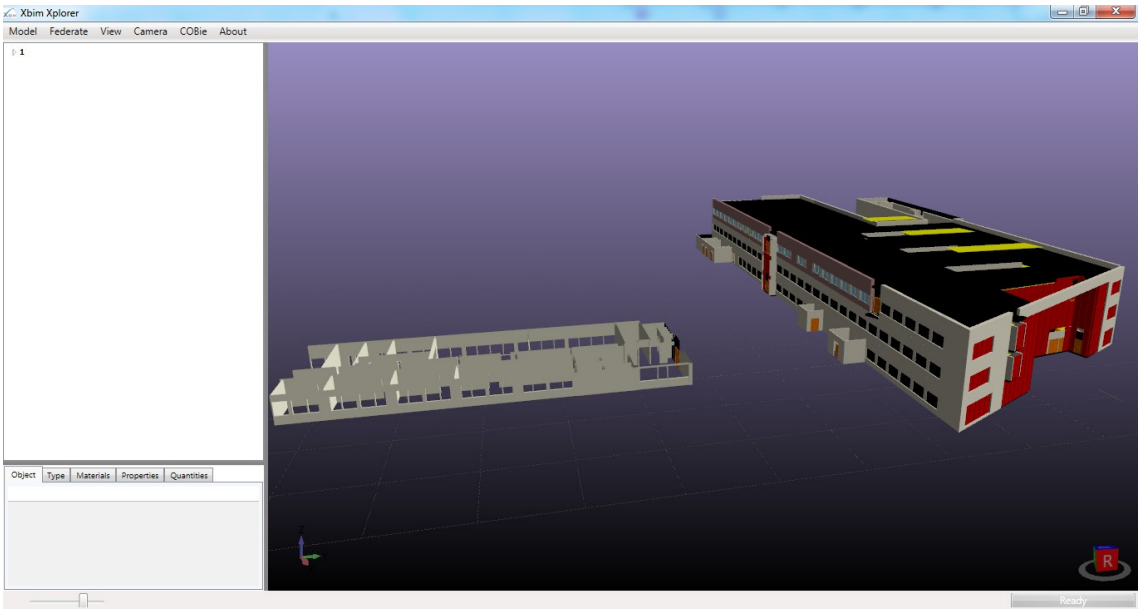
Tilaajan kannalta tietomallit ovat siitä hyödyllisiä, että ne ovat vakioitu toimitustapa suunnitelmien toimittamiseen. Tietomallinnuksen käyttöönotossa Suomi on maailman viiden johtavan maan joukossa. Esimerkiksi suuret infranhaltijat, kuten Liikennevirasto, ovat siirtymässä kokonaan tietomallipohjaiseen tilaamiseen, ja rakennus- ja infratietomallien sovittaminen yhteen voi tuoda uusia mahdollisuuksia rakennettua ympäristöä koskevan tiedon hallintaan. (Savisalo 2012, 25, 41.)

3.2 IFC

IFC (Industry Foundation Classes) on kansainvälinen, avoin standardi oliopohjaisen tiedon siirtoon. IFC:tä kehittää buildingSMART. IFC on vakiintunut pääasialliseksi tiedostotyypiksi tuotemallitiedon siirtoon ohjelmistoista riippumatta CAD-järjestelmien välillä. IFC:tä voidaan siis käyttää ilman uusien ohjelmien asentamista: useimmat alan ohjelmat tukevat sitä jo valmiiksi. Viimeisin versio IFC:stä, IFC4, julkaistiin maaliskuussa 2013 (buildingSMART 2013.).

3.3 Opinnäytetyössä käytetyt IFC:n tarkasteluun ja luontiin käytetyt ohjelmat

Opinnäytetyössä koululta saatuja IFC-malleja tarkasteltiin aluksi käyttämällä avoimen lähdekoodin Xbim Xplorer-ohjelmaa. Xbim Xploreria voidaan käyttää IFC-tiedostojen avaamiseen, niiden navigoimiseen ja tutkimiseen ja tiedostojen viemiseen eri tiedostotyypeissä (esimerkiksi .ifcXML). Jotta tietomallia voisi paremmin esitellä, Xbim Xplorerissa tiettyjä tietomallin elementtejä voidaan väliaikaisesti piilottaa tai niiden läpinäkyvyyttä voidaan säädellä. Kuvassa 9 on esimerkki Xbim Xplorerista. Kuvassa on avattu Saimaan ammattikorkeakoulun uuden rakennuksen IFC-malli, jota voidaan Xbim Xplorerin avulla tarkastella. Tämä tietomalli saatiin koulun tietopankki Haahtelasta.



Kuva 9. IFC-malli Saimaan ammattikorkekoulun tiloista Xbim Xplorerissa

Toinen opinnäytetyössä käytetyistä tietomalleista, Saimaan ammattikorkeakoulun Skinnarilan kampuksen huonetta 6522 (tietomallistudio)vastaava tietomalli, luotiin käyttämällä aiemmin mainittua ArchiCAD-ohjelmaa. ArchiCAD on rakennussuunnittelijan näkökulmasta kehitetty suunnittelijan työkalu, jolla voidaan valmistaa kolmiulotteisia rakennus- ja tietomalleja. Opinnäytetyössä käytetyn huoneen 6522 mallin teki koulun toinen opiskelija, Juha Kemppi. Hän sai muutettua tietomallin suoraan 3D-malliksi ArchiCADissa, jolloin mitään ylimääräistä muutosta ei tarvinnut tehdä. Itse tietomalli, jonka Kemppi loi, olisi kuitenkin voitu toimittaa eteenpäin IFC-tiedostona (M.A.D 2013).

4 Opinnäytetyössä käytetyt 3D-mallinnusohjelmat

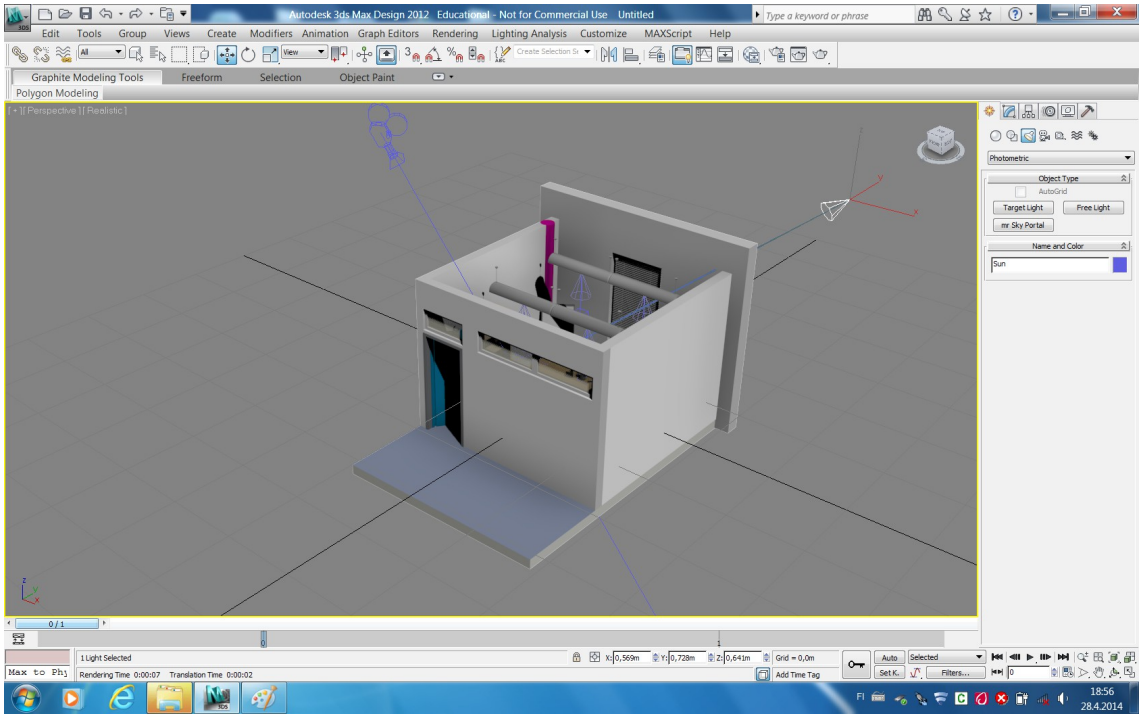
Tässä luvussa esitellään opinnäytetyössä käytetyt 3D-mallinnusohjelmat 3ds Max Design 2012 ja Blender, esitellään niiden käyttöliittymät ja selitetään, miten IFC-tiedosto saadaan 3D-mallinnusohjelmiin.

4.1 Autodesk 3ds Max Design 2012

3ds Max Design 2012 on Autodesk-yrityksen 3ds Max Design -ohjelmiston vuoden 2011 versio. Autodesk aloitti toimintansa 1982 16 työntekijällä. Tällöin yritys kehitti sen tavaramerkiksi muodostuneen AutoCAD-ohjelman. Sen jälkeen Autodesk on laajentunut massiivisesti, ja nykyään Autodesk kattaa kaikki

suunnitteluprosessin vaiheet, 2D-suunnittelusta 3D-mallinnukseen ja myös tietomallien luontiin. Autodesk työstää nykyään 6 600 työntekijää globaalisti ja sen tuotteilla on yli 10 miljoonaa käyttäjää (Autodesk a 2014.).

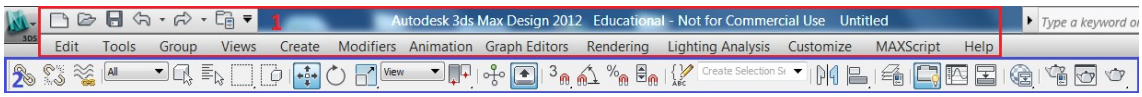
3ds Max Design 2012 on ammattilaiskäyttöön tarkoitettu 3D-mallintamiseen, animointiin ja renderöintiin (kuvan luomista mallista tietokoneohjelman avulla) sekä jälkikäsittelyyn tarkoitettu ohjelma. Tarkemmin määriteltynä 3ds Max Design on hiukan erilainen samalta kuulostavaan 3ds Maxiin. Siinä missä 3ds Max Design sisältää työkaluja arkkitehdeille, suunnittelijoille, insinööreille ja visualisoinnin asiantuntijoille, on 3ds Max suunnattu enemmän pelien valmistajille ja visuaalisiin tehosteisiin sekä liikkuvaan grafiikkaan tai animointiin keskittyville artisteille (Autodesk b 2014). Kuvassa 10 on kuvattu 3ds Max Design 2012 käyttöliittymä. Kuvassa tarkastellaan huoneen 6522 3D-mallia.



Kuva 10. 3ds Max Design 2012 käyttöliittymä.

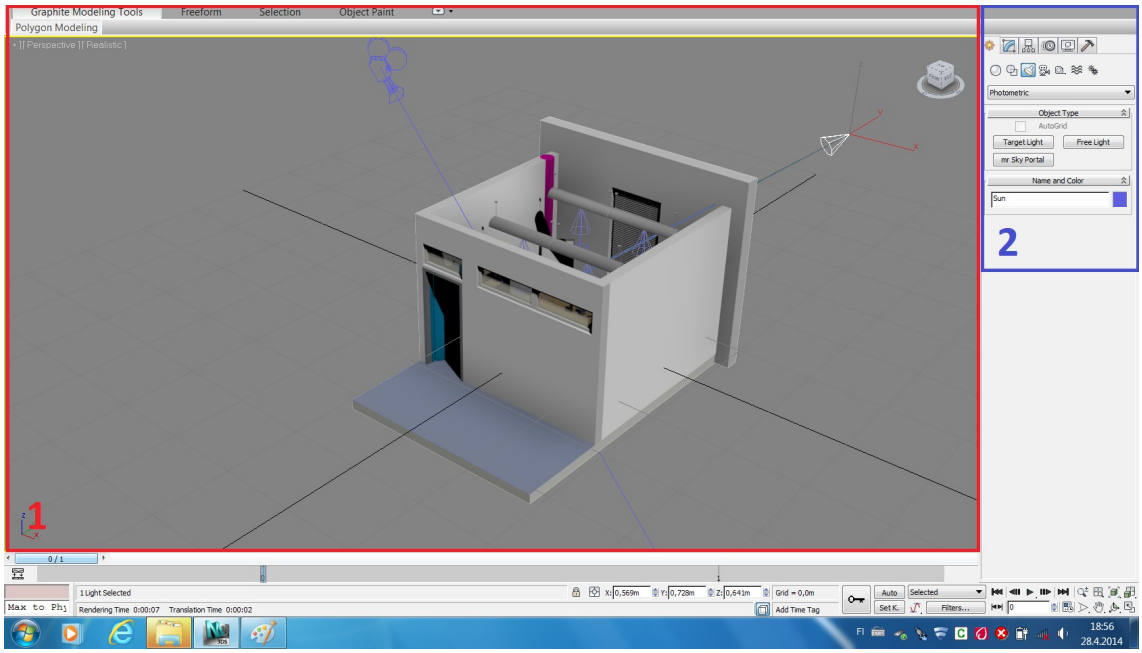
3ds Max Design 2012 käyttöliittymä on jaettu muutamaan tärkeään osaan. Nämä on kuvattu kuvissa 11 ja 12. Ylhäällä on valikkopalkki (Kuva 11, kohta 1). Päätyökalupakki (Kuva 11, kohta 2) sisältää ohjelman peruskomennot. Näkymäikkuna (Kuva 12, kohta 1) tarjoaa katselualueen 3D-tilaan. Käyttöliittymän oikeassa laidassa on komentopaneelit (Kuva 12, kohta 2), jotka

sisältävät pääosan mallintamis- ja animaatiokäskyistä (nämä on jaettu eri välilehtien taakse).



Kuva 11. 3ds Max Design 2012:a käyttöliittymän eri osat (A).

Kuvassa 11 on käyttöliittymän ylempi osa, jossa 1 (ylempi suorakulmio) on valikkopalkki ja 2 (alempi suorakulmio) on päätyökalupakki.



Kuva 12. 3ds Max Design 2012:a käyttöliittymän eri osat (B).

Kuvassa 12 on käyttöliittymän alempi osa, jossa 1 (vasen suorakulmio) on katselualue ja 2 (oikea suorakulmio) ovat komentopaneelit.

Opinnäytetyössä 3dsMax Design 2012:ta ei käytetty kuin avaamaan alkuperäinen IFC-tiedosto ja muuttamaan se Unityn tunnistamaan tiedostomuotoon. Tämän takia tässä raportissa ei keskitytä tarkasti kuvaamaan 3ds Max Design 2012:n erinäisiä 3D-mallinnukseen käytettyjä ominaisuuksia ja toimintoja. Kuvan tietomallistudion 3D-malli oli jo valmiiksi 3D-malli, joten sen pystyi siirtämään suoraan Unityyn. Kuvassa 9 nähty ammattikorkeakoulun tietomalli sen sijaan piti ensin saada 3ds Max Design 2012:a, jossa se piti vielä

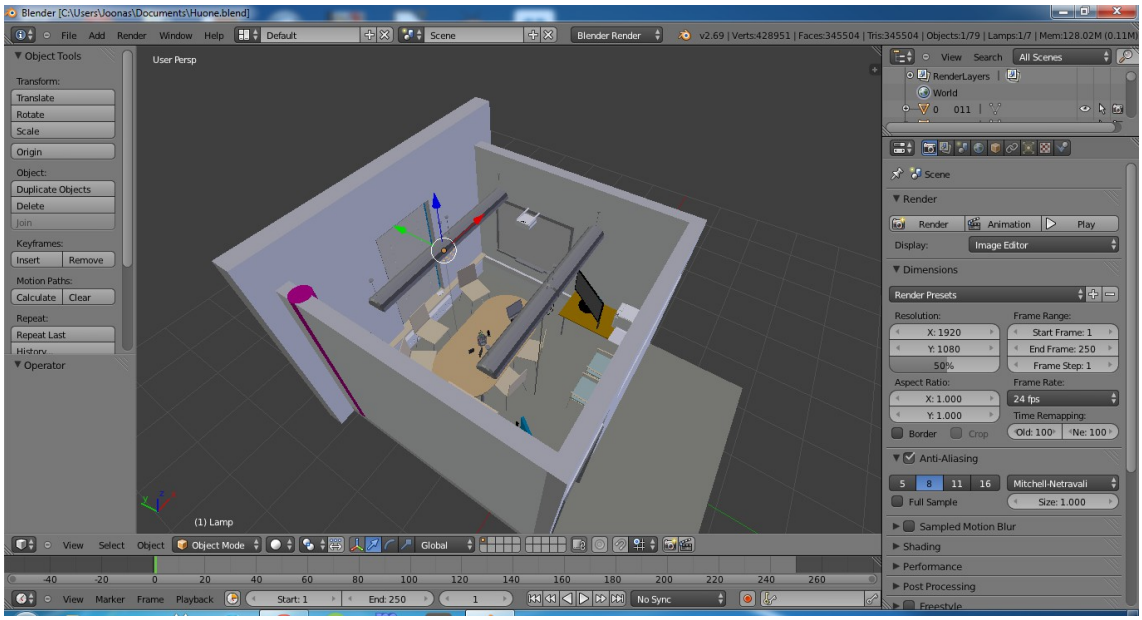
muuttaa Unityn tunnistamaan tiedostomuotoon. Tämä prosessi kuvataan tarkemmin luvussa 5.

4.2 Blender

Blender on avoimen lähdekoodin 3D-mallinnusohjelma, joka on tarkoitettu kolmiulotteisen grafiikan mallinnukseen, renderöintiin, animointiin ja jälkikäsittelyyn. Ohjelma on täysin ilmainen ja avoimen lähdekoodin takia kaikkien muokattavissa.

Blenderin historia alkaa vuodelta 1988, jolloin Ton Roosendaal oli mukana perustamassa Alankomaista animaatiostudio NeoGeota, josta tuli pian Alankomaiden suurin animaatiostudio. Blender syntyi tässä animaatiostudiossa vuonna 1995. Vuonna 1998 Roosendaal perusti uuden yrityksen nimeltä ”Not a Number” (NaN), jonka tarkoituksena oli entisestään kehittää ja markkinoida Blenderiä. Blender esiteltiin ensimmäisen kerran vuonna 1999. Lupaavasta alusta huolimatta Blender ei kuitenkaan tahtonut löytää paikkaansa markkinoilla, ja se melkein kuolikin kaksi kertaa vuosina 2000 - 2002. NaN ajautui konkurssiin ja Blenderin kehitys pysähtyi, mutta Blenderin käyttäjäkunnan kannustamana Roosendaal perusti Blender-säätiön, jonka tarkoituksena oli jatkaa Blenderin kehitystä ja promootiota vapaan lähdekoodin ohjelmana. ”Free Blender”-kampanjan avulla Roosendaal kumppaneineen keräsi tarvittavat varat ostaa Blenderin oikeudet, ja 13. lokakuuta 2002 Blender julkaistiin GNU-lisenssin alaisena. (Blender 2013.)

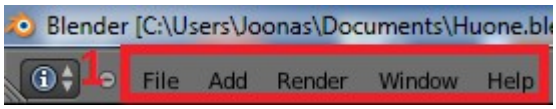
Opinnäytetyössä käytettiin Blenderin toiseksi uusinta versiota, Blender 2.69 (uusin versio, 2.70a julkaistiin huhtikuun 11. 2014). Kuvassa 13 on kuvattu Blenderin käyttöliittymä.



Kuva 13. Blender 2.69:n käyttöliittymä.

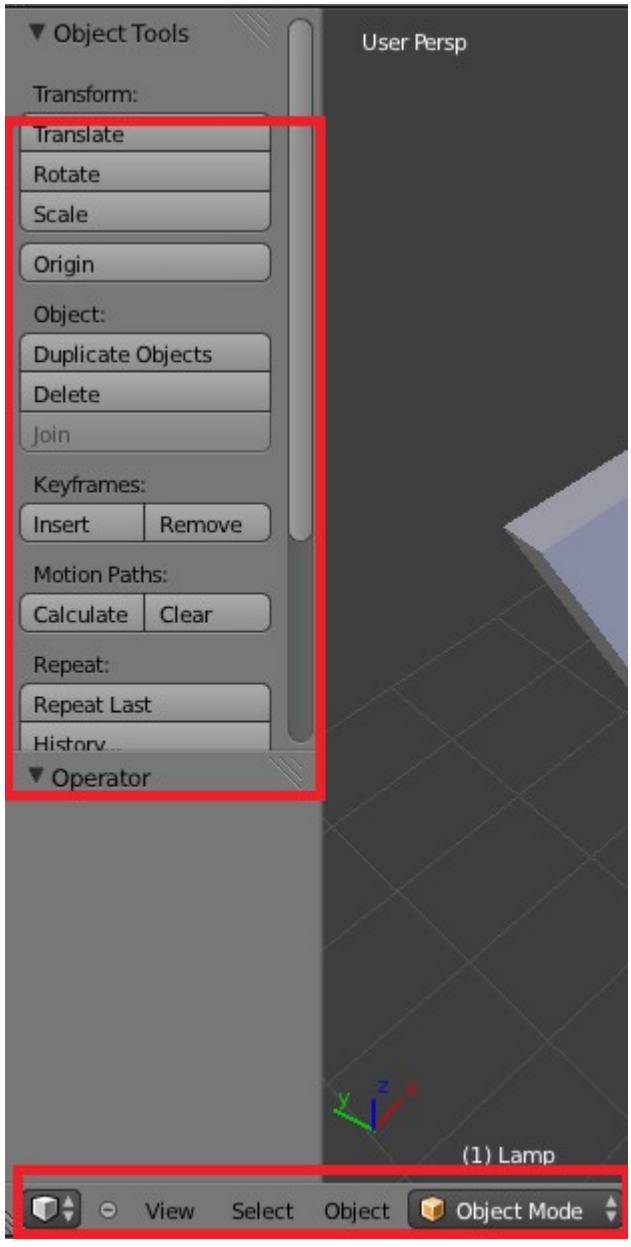
Blenderin käyttöliittymä jakaa muutamia osia 3ds Max Design 2012 käyttöliittymän kanssa. Näitä ovat:

- valikkopalkki (Kuva 14)
- päätyökalupalkki, jolla hallitaan ja manipuloidaan tiettyä objektia ja valikko, josta valitaan päätyökalupalkin toiminto (Kuva 15)
- näytettävän näkymän valinta (Kuva 16, kohta 1)
- näytettävän kohtauksen tai kentän valinta (Kuva 16, kohta 2)
- kuvan renderöintiin käytettävän renderoijan (työkalu, joka suorittaa renderöinnin) valinta (Kuva 16, kohta 3)
- katselunäkymä (Kuva 16, kohta 4)
- erinäisiä objektin hallintaan käytettäviä toimintoja sekä animoinnissa käytettävät painikkeet (Kuva 16, kohta 5)
- kohtauksessa olevien objektien lista (Kuva 17, kohta 1)
- ja komentopaneelit (Kuva 17, kohta 2) jotka sisältävät pääosan mallintamis- ja animaatiokäskyistä (nämä on jaettu eri välilehtien taakse).



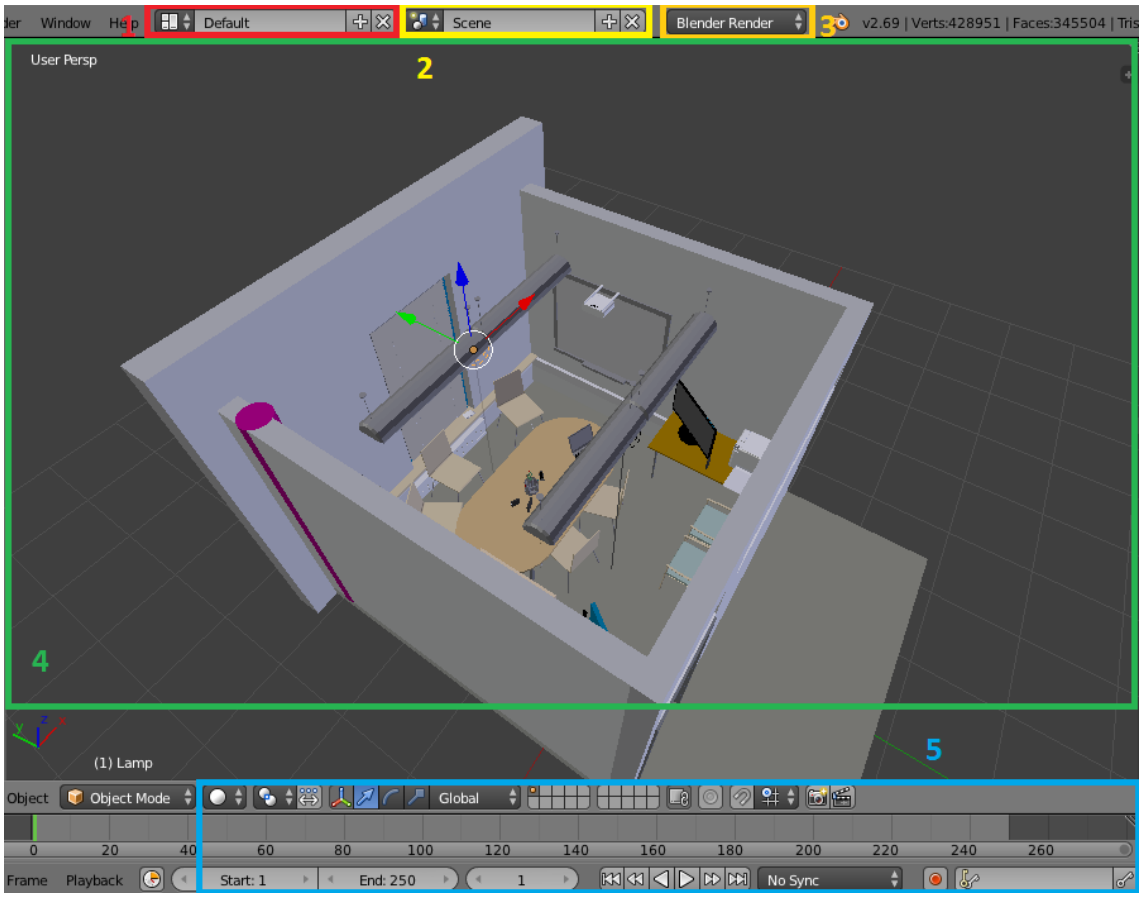
Kuva 14. Valikkopalkki

Valikkopalkki sisältää Blenderin käytön yleisimmät toiminnot.



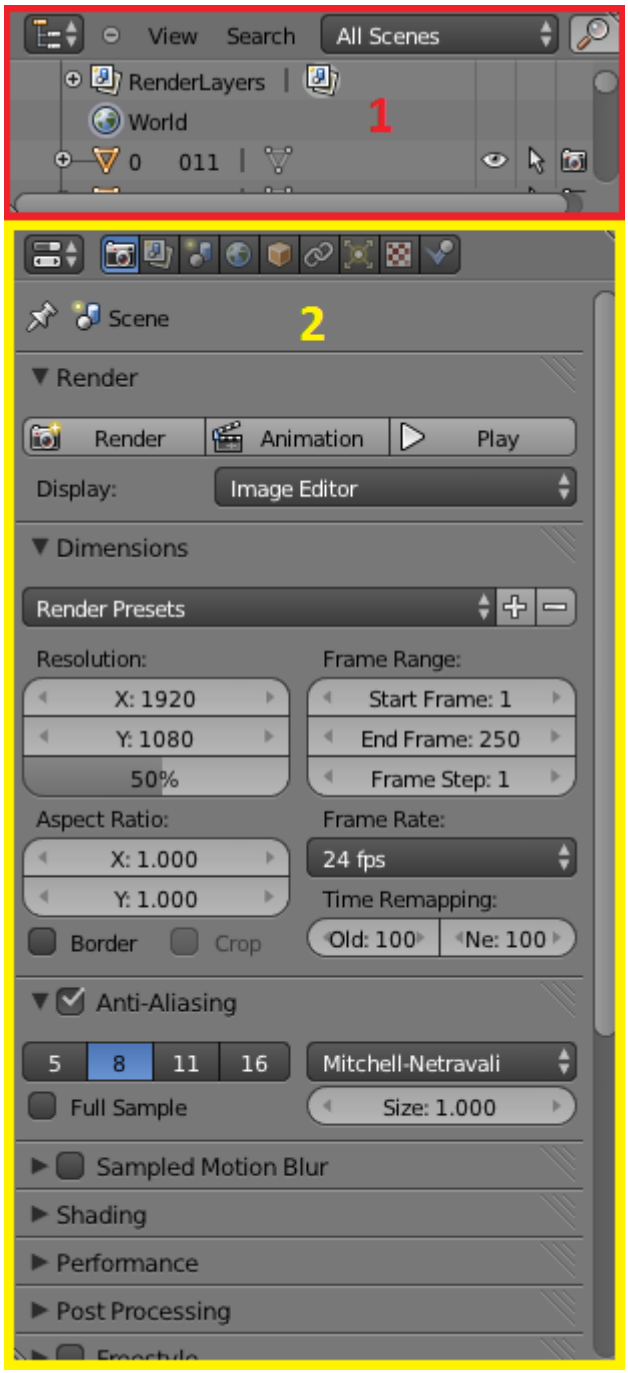
Kuva 15. Päätyökalupalkki ja valikko, josta vaihdetaan sen toimintoja

Tässä tapauksessa päätyökalupalkilla hallitaan objekteja. Valikko, jossa tässä kuvassa lukee ”Object Mode”, vaihtaa päätyökalupalkin toimintoja.



Kuva 16. Blender 2.69:n käyttöliittymän eri osat (A)

Kuvassa on näytettävän näkymän valinta 1 (ylin vasemmanpuoleisin värillinen suorakulmio), näytettävän kohtauksen valinta 2 (ylin keskimmäinen värillinen suorakulmio) renderoijan valinta 3 (ylin oikeanpuoleisin värillinen suorakulmio), katselualue 4 (suuri suorakulmio keskellä) ja objektin hallintaan käytettäviä toimintoja sekä animoinnissa käytettävät painikkeet 5 (alin suorakulmio).



Kuva 17. Blender 2.69:n käyttöliittymän eri osat (B)

Kuvassa on Blenderissä sillä hetkellä käsiteltävänä olevassa kohtauksessa olevien objektien lista 1 (ylempi suorakulmio) ja komentopaneelit 2 (alempi suorakulmio)

Kuten käytettäessä 3ds Max Design 2012:ta, käytettiin Blenderiä opinnäytetyössä lähinnä muuttamaan toinen tietomalli Unityn tunnistamaksi 3D-

malliksi. Tämän takia tässä raportissa ei keskitytä tarkasti kuvaamaan Blenderin erinäisiä 3D-mallinnukseen käytettyjä ominaisuuksia ja toimintoja.

4.3 IFC-tietomalli 3ds Maxiin ja Blenderiin

3ds Max Design 2012 ja Blender eivät suoraan tunnista IFC-tiedostoja, joten niitä varten täytyi molempiin ohjelmiin ensin asentaa erillinen liitännäinen, ifcOpenShellin tarjoamat IfcMax ja IfcBlender. Molemmat löytyvät seuraavasta osoitteesta:

http://ifcopenshell.org/

IfcOpenShell muuntaa IFC-tiedostojen epäsuoran geometrian tarkaksi geometriaksi, jota mikä tahansa CAD-ohjelmisto tai mallinnusohjelma voi ymmärtää ja hyödyntää (IfcOpenShell 2014).

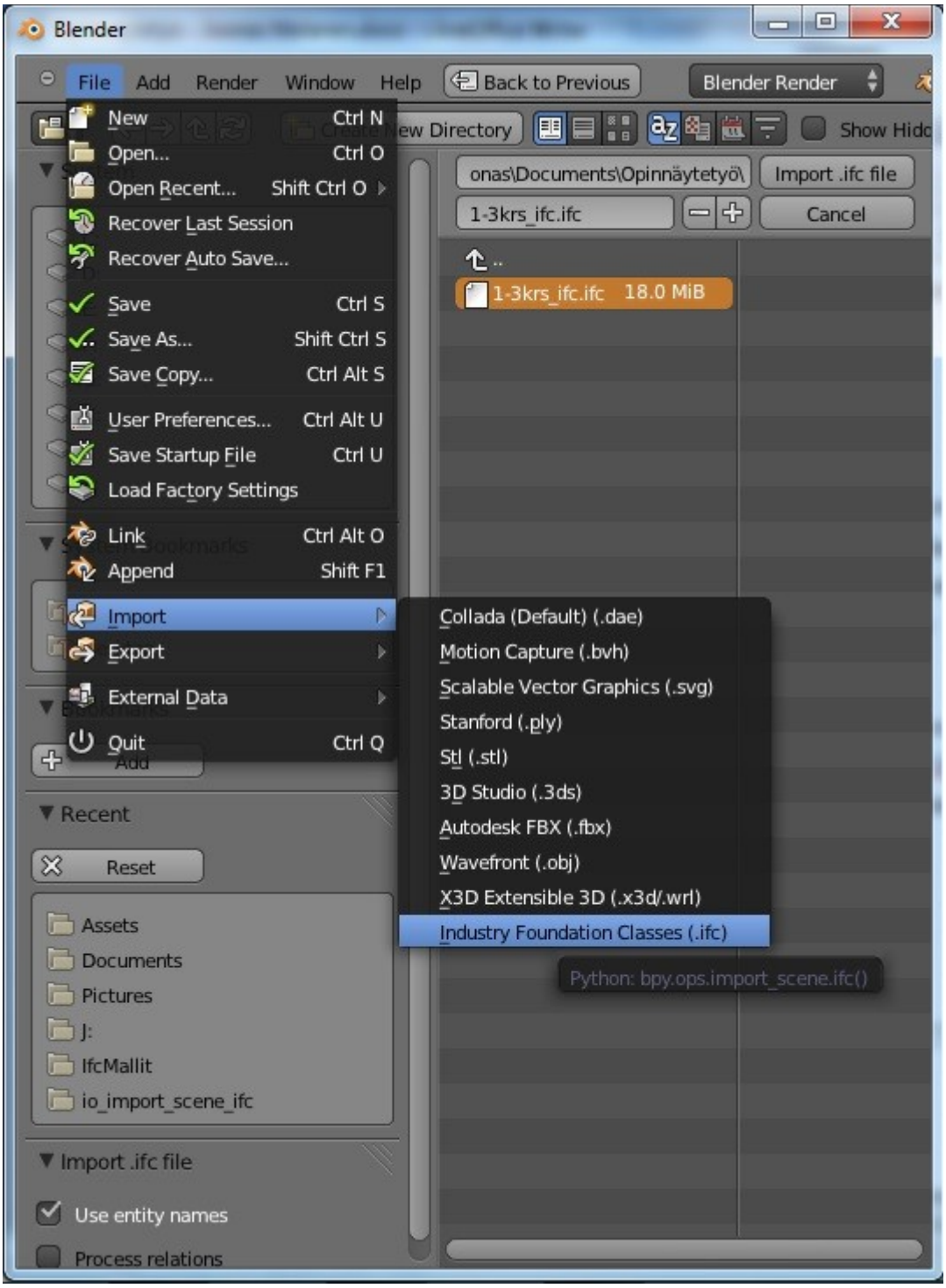
Nämä täytyi asentaa koneille ennen kuin IFC-tiedostoja pystytään käyttämään 3D-mallinnusohjelmissa. Koska 3ds Max Designia käytettiin koululta lainatulta työkoneella, ei siihen ollut oikeuksia suoraan asentaa mitä tahansa. Liitännäinen täytyi käydä asennuttamassa Saimaan ammattikorkeakoulun IT-tukihenkilöllä. Blenderiin samanlaisia rajoituksia ei ollut, joten siihen ohjelma asennettiin ilman ongelmia. Molemmissa ohjelmissa liitännäinen vain ladattiin Internetistä ja kopioitiin tai purettiin ohjelmien liitännäiskansioon. Blenderissä liitännäinen täytyi vielä erikseen ottaa käyttöön erillisestä valikosta.

5 Työn suoritus

Tässä luvussa kuvataan opinnäytetyön suoritus eri vaiheineen.

5.1 3D-mallien siirtäminen Unityyn

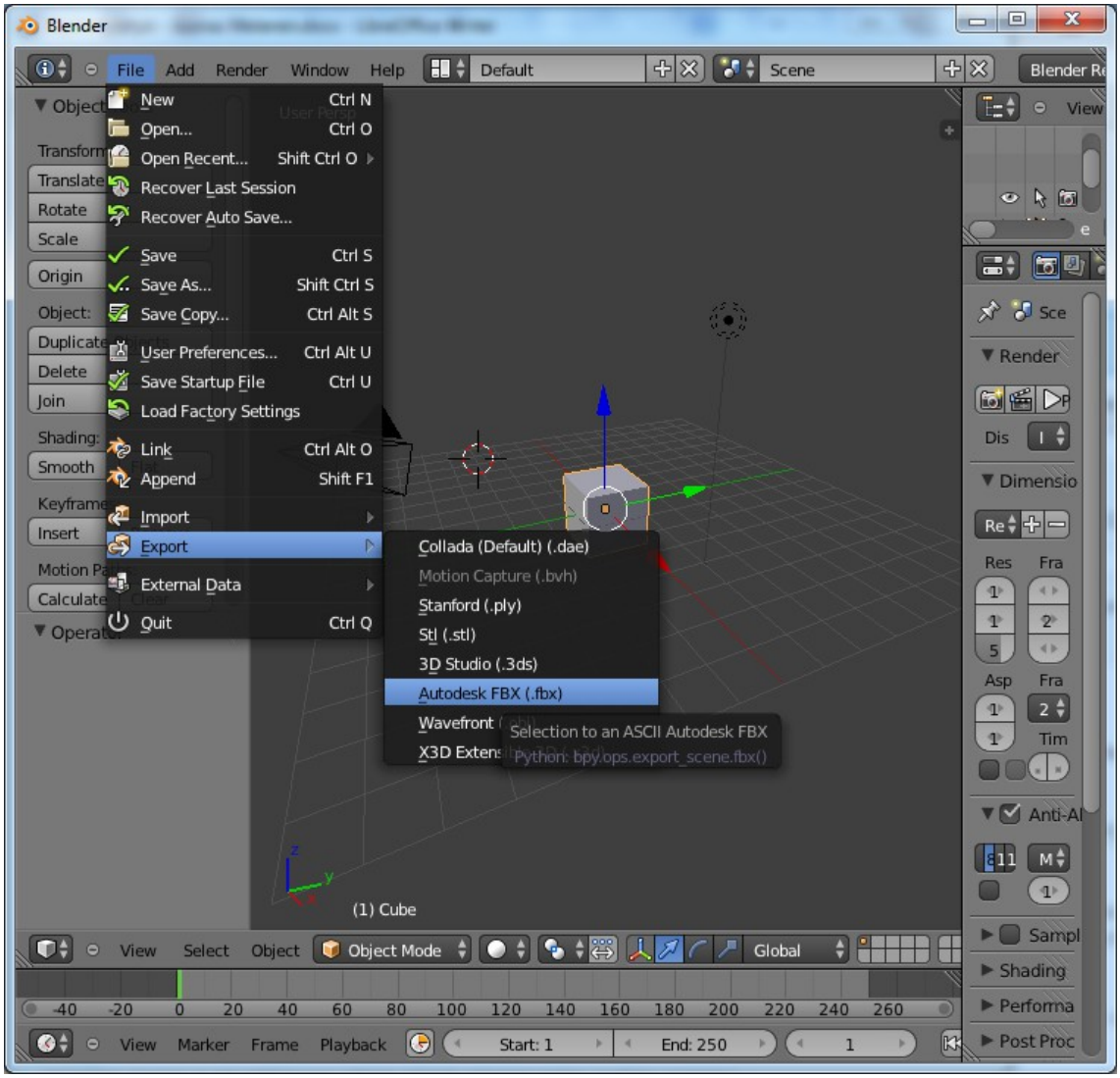
Aluksi tietomalli piti saada yhteen 3D-mallinnusohjelmista. Tämä onnistui ongelmitta kun IfcOpenShell-liitännäiset oli ensin asennettu. Sekä 3ds Max Designista että Blenderistä valittiin vain ”Import” eli ”Tuo” ja tiedostotyypiksi valittiin IFC. Tämän jälkeen vain valittiin työssä käytettävä IFC-tiedosto ja tuotiin se 3D-mallinnusohjelmaan. Tämän prosessin suoritus Blenderissä kuvataan kuvassa 18.



Kuva 18. IFC-tiedoston tuominen Blender-3D-mallinnusohjelmaan.

Kun kuva on tuotu 3ds Max Designiin tai Blenderiin, täytyy ne seuraavaksi saada Unityyn. Unity tukee useita tiedostotyyppejä, mutta tätä opinnäytetyötä varten 3ds Max Designista IFC-tiedosto vietiin Unityyn .fbx-tiedostona. Blenderin oma tiedostotyyppi .blend toimii myös sellaisenaan Unityssa. Sitä ei

tarvitse muuttaa mihinkään toiseen tyyppiin. Kuvassa 19 näytetään IFC-tiedoston vieminen Blenderistä. Mallin voi tallentaa mihin haluaa.



Kuva 19. 3D-mallin vieminen Blenderistä Unityyn.

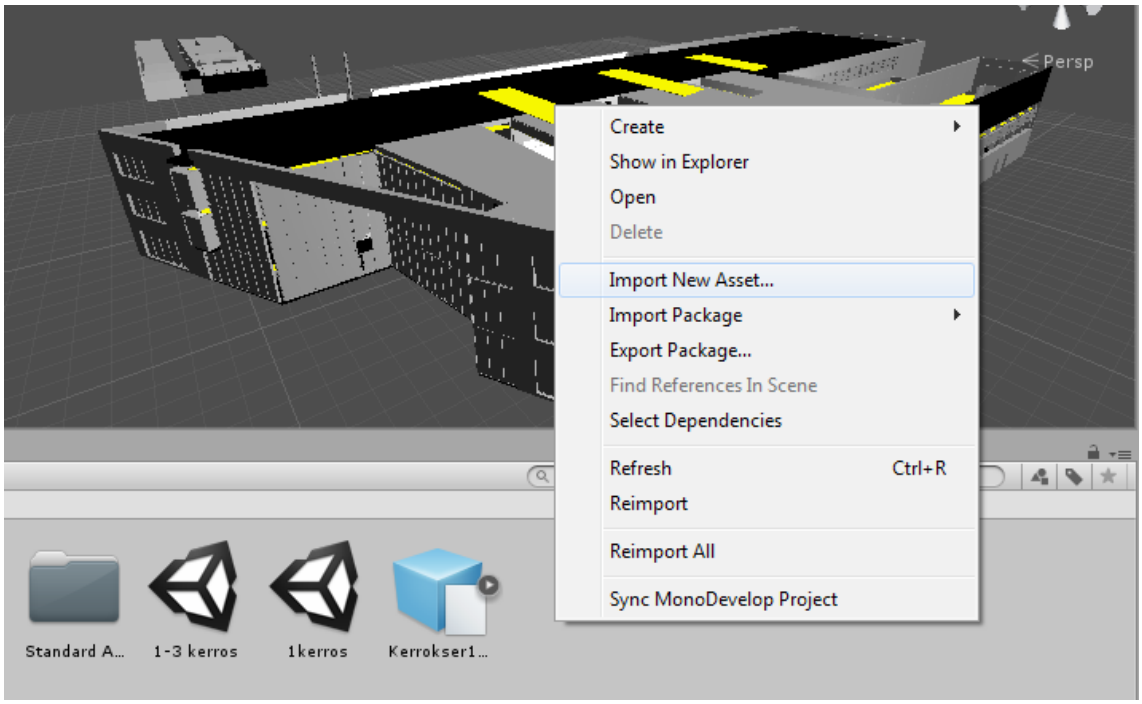
Sama prosessi toimisi myös muille tietomalleille, mutta tässä opinnäytetyössä testattu toinen tietomalli tietomallistudiosta (Saimaan ammattikorkeakoulun Skinnarilan kampuksen huone 6522) oli jo valmiiksi 3D-mallina (tiedostotyyppinään .3ds), jonka Unity tunnistaa, joten sitä ei tarvitse käydä erikseen viemään eri tiedostotyyppinä.

5.2 3D-mallien käyttäminen ympäristöinä Unityssa

Kun tietomallit on saatu vietyä Unityn tunnistamissa tiedostotyypeissä, ne täytyy ottaa käyttöön Unityssa. Kenttä ja muut Unityssa käytettävät 3D-mallit ovat

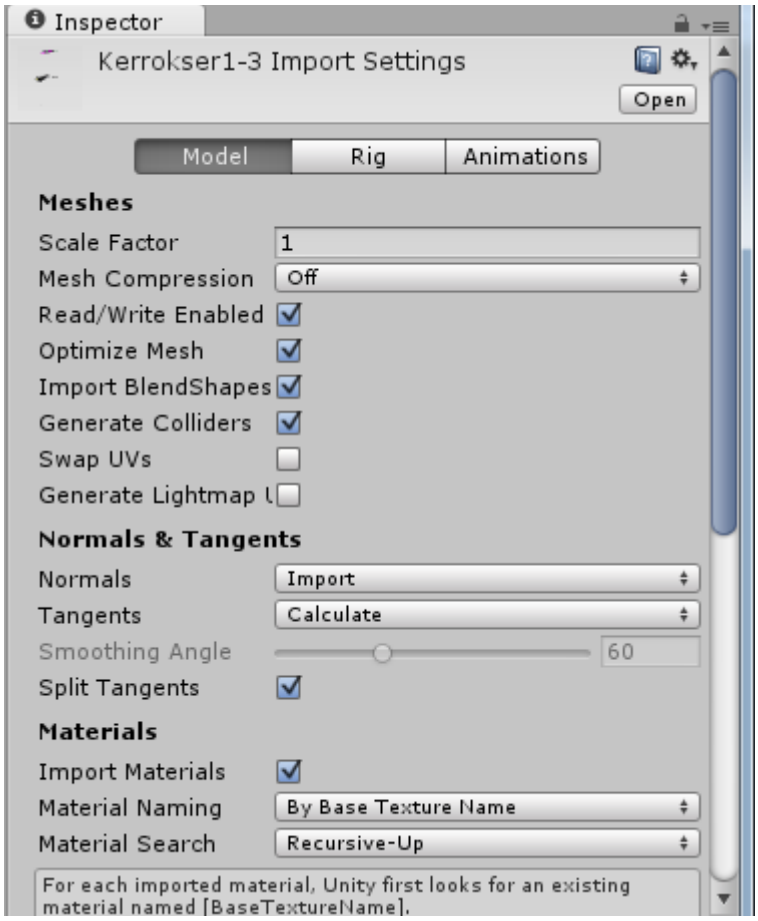
”Assetteja” eli käytettäviä voimavaroja. Jotta Unity tunnistaa objektin tai 3D-mallin Assetiksi, täytyy se ensin joko tallentaa Unity-projektin ”Assets”-kansioon tai valita itse projektissa ja sen tasossa ”Import New Asset”, joka avaa tavallisen Windowsin valintaikkunan. Tässä ikkunassa sitten valitaan haluttu 3D-malli tai muu objekti tai Asset.

Kun tietomallin 3D-malli on saatu Unityyn, se otetaan käyttöön raahaamalla se Unityn tason näkymään. Tätä ennen on tosin luotava mallille ”Colliderit” eli törmäyttimet. Tämä varmistaa, että 3D-malli on kiinteä eikä pelaajan hahmo vain tipu lattiasta läpi heti käynnistäessään peliä. Lisäksi tätä opinnäytetyötä varten 3D-mallin skaalaa oli kasvatettava hiukan, jotta pelaajan hahmo mahtuisi liikkumaan siellä. Kuvissa 20 ja 21 näytetään kaikki nämä toiminnot.



Kuva 20. Uuden Assetin tuominen Unityyn.

Halutun tiedoston voi myös tallentaa Unityn projektin Assets-kansioon, jos Unity tukee tiedostotyyppiä.



Kuva 21. Tuodun 3D-mallin asetukset.

Vaikka 3D-malli koostuisi useista osista, toimii se silti myös yhtenä objektina. Täten 3D-mallia voidaan muokata joko kaikkia sen osia kerralla muuttamalla tai muuttamalla vain yhtä sen osaa (esimerkiksi huoneen lattia).

5.3 Ympäristöjen ulkoasuun tehtävät muutokset

Koska tietomalleissa määritelty ympäristöjen valaistus ei ollut siirtynyt sellaisenaan Unityyn, täytyi molempiin ympäristöihin lisätä uusia valoja. Ilman valoja ympäristö olisi pilkkopimeä.

Unityssa valoja lisätään valitsemalla Unityn valikkopalkin kohdasta **GameObject → Create Other**. Tämän jälkeen tarjolla on neljä eri valoa (tässä opinnäytetyössä kolme, sillä viimeinen valoista, Area Light, on vain Unity Prossa):

- **Suuntavalo** (Directional Light). Suuntavalo loistaa yhteen suuntaan tasaisesti kaikkialle kenttään. Sitä käytetään pääasiassa simuloimaan auringon tai vastaavan tuottamaa valoa. Molemmissa kentissä on yksi Suuntavalo kaukana valaisemassa koko kenttää.

- **Pistevalo** (Point Light). Pisteen muotoinen valo, joka loistaa valoa tasaisesti kaikkiin suuntiin. Näitä valoja käytettiin opinnäytetyössä eniten ympäristön valaisemiseen.

- **Kohdevalo** (Spot Light). Kartion muotoinen valo, joka loistaa yhteen suuntaan. Käytetään esimerkiksi taskulamppuja tehdessä.

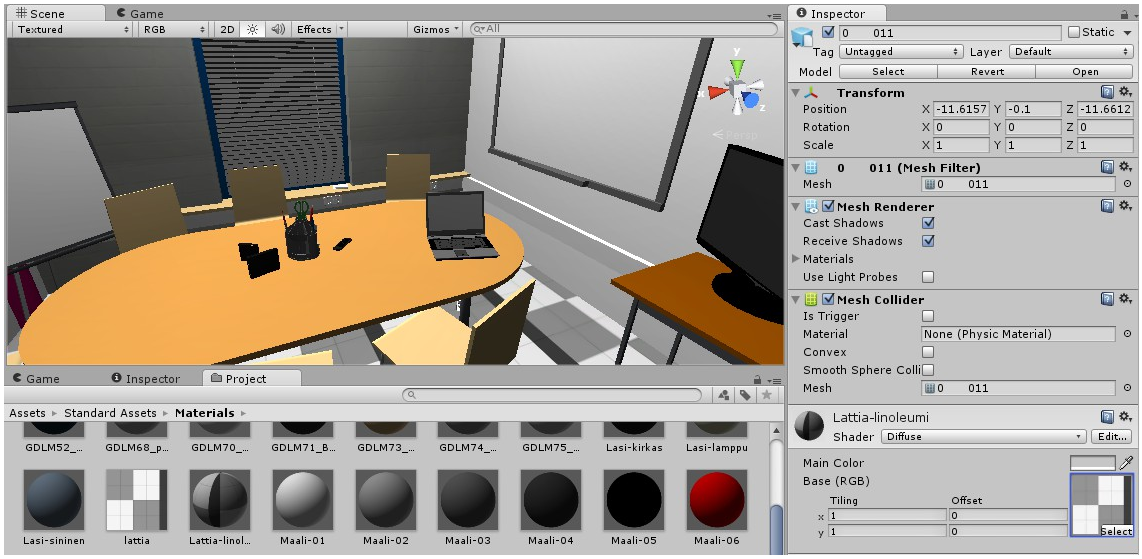
- **Aluevalo** (Area Light). Valoa, joka lähtee neliön muotoisen tason toiselta puolelta tasaisesti. Nämä valot ovat liian raskaita järjestelmälle, joten niitä ei käytetä reaaliaikaisesti.

Saimaan ammattikorkeakoulun uutta rakennusta kuvaavaan ympäristöön lisättiin yksi suuntavalo kauas mallin ulkopuolelle valaisemaan ympäristöä hellästi. Tämän lisäksi mallin sisälle lisättiin kolme pistevaloa eri kohtiin valaisemaan aluetta mahdollisimman laajasti. Saimaan ammattikorkeakoulun tietomallistudiota kuvaavaan ympäristöön lisättiin neljä pistevaloa ja yksi suuntavalo kauas. Valojen intensiivisyyttä voidaan säätää yksinkertaisella liukurilla.

Pelaaja voi vaikuttaa tietomallistudion valoihin itse: painamalla E-näppäintä valokatkaisimen vieressä, valot voi sammuttaa ja laittaa takaisin päälle.

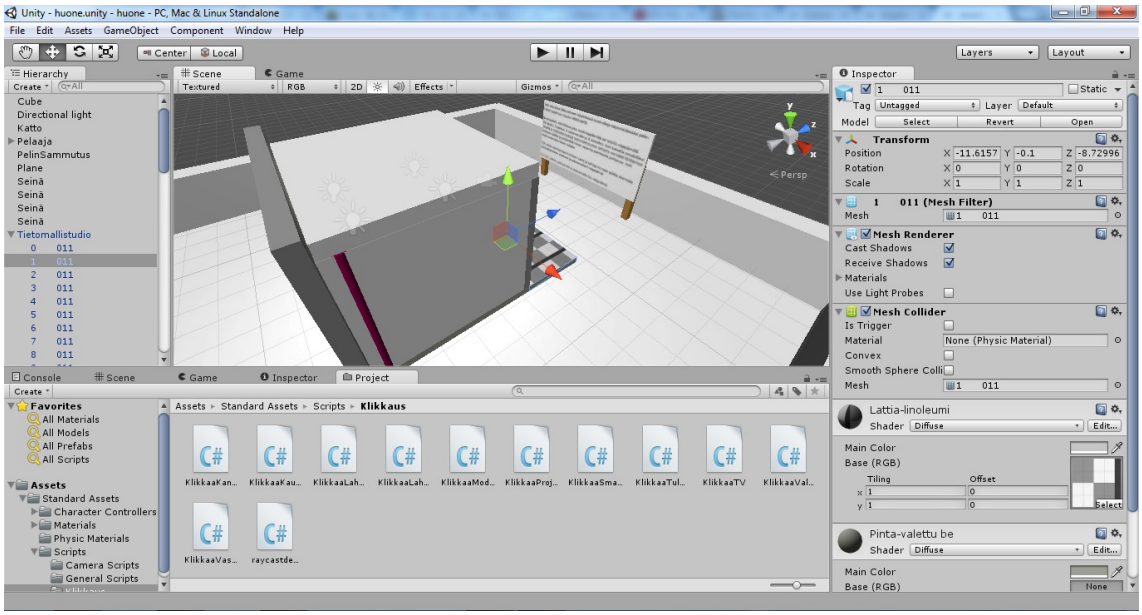
Saimaan ammattikorkeakoulun tiloihin ei tehty mitään ulkoasullisia muutoksia, mutta tietomallistudioon tehtiin kaksi. Huoneen tiiliseinä ja lattia olivat menettäneet niille tietomallia tehdessä määritellyt materiaalit ja pinnat, joten nämä korvattiin Timo Lehtoviidan pyynnöstä tekstuureilla; tiiliseinälle tiilitekstuuri, lattialle sen kuviointia vastaava tekstuuri. Tekstuuri liitettiin objekteihin ensin tuomalla se uudeksi Assetiksi ”Import New Asset”-menetelmällä ja sen jälkeen vetämällä uusi tekstuuri objektiin määrätyn materiaalin tekstuuriksi. Kuva 22 havainnollistaa tätä. Kuvassa ollaan Materials-

kansiossa, jossa lattia-tekstuuri on liitetty huoneen lattian Lattia-linoleumi-materiaaliin tekstuuriksi.



Kuva 22. Lattian tekstuuri

Tämän lisäksi tietomallistudion ympäristöön lisättiin aitaus, jottei pelaaja vahingossa tipu pelialueen laitojen ulkopuolelle ja joudu aloittamaan alusta. Huoneelle lisättiin katto, joka aiemmin puuttui, ja ympäristöön lisättiin ohjekyltti, josta pelaaja voi lukea pelihahmon ohjaamiseen tarkoitetut kontrollit ja saa idean pelin tarkoituksesta. Kuvassa 23 on esitelty nämä muutokset.



Kuva 23. Tietomallistudion ympäristön muutokset Unityssa.

Jos 3D-mallin pohjana toimivaan tietomalliin tehdään muutoksia 3D-mallin käyttöönoton jälkeen ja jos tätä uutta tietomallia halutaan käyttää ympäristönä pelissä, on aluksi uudesta tietomallista tehtävä uusi 3D-malli. Jos muutokset ovat pieniä, esimerkiksi yksi ylimääräinen tuoli (joka on samanlainen kuin huoneessa jo olevat tuolit) huoneessa, nämä muutokset voidaan tehdä joko valitussa 3D-mallinnusohjelmassa tai Unityssa muokkaamalla jo käytössä olevaa 3D-mallia. Tällöin uuden 3D-mallin luomiselle ei ole tarvetta. Jos uusi tietomalli on kuitenkin huomattavasti erilainen verrattuna vanhaan (esimerkiksi kokonaan uusi ympäristö), täytyy siitä tehdä uusi 3D-malli.

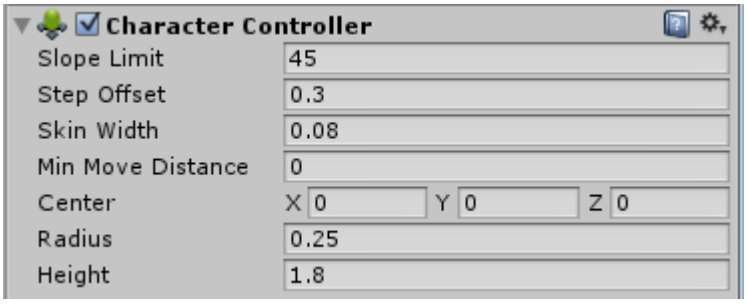
Tällöin koko tietomallin Unityyn saattava prosessi tehdään kokonaan alusta uudelleen. Uudesta tietomallista luodaan uusi 3D-malli, joka siirretään Unityyn. Unityssa uudesta 3D-mallista tehdään uusi peliympäristö ja sille luodaan törmäyttimet ja rajat (jotta pelaaja ei vahingossa tipu alueelta). Ympäristöön voidaan lisätä uudet valot tai olemassa olevat valot voidaan siirtää niiden uusille paikoilleen. Myös pelaaja itse voidaan joko tehdä uudelleen tai siirtää uudelle paikalleen peliympäristössä. Ympäristössä käytetyt tekstuurit kiinnitetään uudelleen niitä tarvitseviin objekteihin tai pintoihin (tekstuurit voidaan käyttää uudelleen sellaisinaan, jos pinta tai objekti johon se kiinnitetään on sama). Aiemmin luodut skriptit voidaan myös käyttää uudelleen sellaisinaan, kunhan ne kiinnitetään oikeisiin peliobjekteihin ja kunhan ne ja niitä vastaavat peliobjektit nimetään oikein.

Kaikki uuden tietomallin käyttöönottoon tarvittavat toimenpiteet on lueteltu liitte 1:ssä.

5.4 Liikuteltavan hahmon lisääminen peliympäristöön

Ympäristössä hahmon silmistä ohjatun hahmon luonti lähtee liikkeelle hahmon luomisesta peliin. Tämä tehdään luomalla uusi tyhjä peliobjekti valitsemalla **GameObject → Create Empty** ja liittämällä siihen hahmon ohjaukseen tarkoitettu ”Character Controller” eli hahmon ohjaaja valitsemalla **Component → Physics → Character Controller**. Character Controller määrää pelihahmolle valmiiksi tiettyjä ominaisuuksia, kuten kuinka jyrkkää mäkeä hahmo voi nousta,

miten korkea hahmo on ja mikä sen säde on ylhäältä katsottuna. Character Controller ominaisuuksineen on esitelty kuvassa 24.



Kuva 24. Character Controller

Kuva 24 on otettu tietomallistudion Character Controllerista. Tässä hahmon säde eli Radius on 0,25, koska hahmo oli aluksi liian leveä mahtuakseen liikkumaan tietomallistudiossa.

Kun Character Controller on luotu, täytyy kamera saada pysymään hahmon silmissä. Haluttu kamera (tässä tapauksessa pääkamera, koska hahmon silmistä kuvatussa pelissä ei ole muita kameroita häiritsemässä) asetetaan täsmälleen samaan kohtaan kuin pelihahmo ja asetetaan pelihahmon lapseksi. Tämä onnistuu vetämällä pääkamera Unityn ”Hierarchy”-eli hierarkia-palkissa pelihahmon sisään, jolloin kamerasta tulee pelihahmon lapsi. Mitä muutoksia pelihahmolle tapahtuukaan, ne muutokset vaikuttavat myös kameraan.

Tämän jälkeen hahmo täytyy saada liikkumaan. Tämä onnistuu tekemällä erillinen skripti hahmon liikuttamista varten. Skripti voidaan tehdä mihin tahansa projektin Assets-kansioista, mutta järjestelmällisyyden kannalta ne olisi hyvä koota yhteen paikkaan. Skripti luodaan valitsemalla **Assets** → **Create**, jonka jälkeen valitaan haluttu ohjelmointikieli tai painamalla hiiren oikeaa painiketta halutussa kansiossa ja valitsemalla aukeavasta valikosta **Create** → haluttu ohjelmointikieli. Tässä opinnäytetyössä ohjelmointi suoritettiin pääasiassa C#-ohjelmointikielellä, mutta pelin tähtäin ja siinä toimiva zoomaus tehtiin Javascript -ohjelmointikielellä.

Hahmon ohjaamisen tarkoitettu skripti ohjelmoitiin opinnäytetyössä käyttäen Unityn mukana tullutta MonoDevelop 4.01:tä. Ohjelmointikielenä toimi C#.

Skriptiin on valmiiksi ohjelmoitu Start- ja Update-metodit. Start-metodia käytetään skriptin alustukseen, Update-metodia kutsutaan pelin ollessa käynnissä joka sekunti. Tämä tarkoittaa että Updatea kutsutaan aina.

Aluksi skriptiin on lisättävä vaatimus Character Controllerille.

```
[RequireComponent(typeof(CharacterController))]
```

Tämän jälkeen määritellään skriptissä liikkumiseen ja ympärille katsomiseen tarkoitetut muuttujat. Muuttujat ovat julkisia, jotta muut voivat käyttää niitä. Ne ovat float-muuttujia, koska float on int-muuttujaa huomattavasti joustavampi.

```
public float movementSpeed = 5.0f;
public float mouseSensitivity = 5.0f;
float verticalRotation = 0;
public float upDownRange = 60.0f;
public float verticalVelocity = 0;
```

MovementSpeedia käytetään laskemaan hahmon liikkumisnopeus, mouseSensitivitylla lasketaan hiiren liikkumisnopeus, verticalRotationia käytetään hahmon katselukulman rajoittamseen, upDownRange kuvaa hahmon maksimi- ja minimi katselukulman ja verticalVelocity hahmon putoamisnopeuden. Lisäksi alussa alustetaan CharacterController-olio characterController.

CharacterController characterController;

Tämän jälkeen hiiri on lukittava ruudun keskelle. Koska pelaajan täytyy pystyä klikkaamaan pelissä olevia objekteja näkemättä hiirtä, täytyy hiiren pysyä ruudun keskellä. Tämä tehdään kertomalla skriptille hiiren lukitsemisen olevan totta. Samalla Start-metodissa pyydetään characterControlleria hakemaan Character Controller-komponenttia tarkistaakseen, että se todella on objektissa, johon skripti liitetään.

```
void Start () {
Screen.lockCursor = true;
characterController = GetComponent<CharacterController>();
          }
```

Update-metodissa, jota kutsutaan joka ruutu, ruudun lukitseminen määritellään uudelleen. Tämä varmistaa, ettei hiiri juutu alussa annettuun paikkaan. Tämän jälkeen suoritetaan hahmon kääntäminen hiirellä. Muuttuja rotLeftRight ottaa hiiren X-tason eli horisontaalisen liikkeen ja kertoo sen hiiren herkkyydellä. Tämän jälkeen transform.Rotate muuttaa hahmon asentoa ja kääntää sitä annetun rotLeftRightin mukaan.

```
//Kääntäminen
float rotLeftRight = Input.GetAxis("Mouse X") * mouseSensitivity;
transform.Rotate (0,rotLeftRight,0);
```

Sama toimii myös ylös ja alas katseltaessa: muuttuja verticalRotation ottaa hiiren Y-akselin eli vertikaalisen liikkeen ja kertoo sen hiiren herkkyydellä. Tämän jälkeen verticalRotationia rajoitetaan: ihmisen niska ei käänny täyttä 180 astetta eikä todellakaan voi pyöriä itsensä ympäri vertikaalisesti. Clamp rajoittaa sen liikettä ottamalla alkuperäisen verticalRotationin ja rajoittamalla sitä ylä- ja alalaidoista. Lopuksi pelihahmon kameran sijaintia muokataan hiiren liikkeiden ja verticalRotationin mukaan. Koodissa mainittu Quaternion.Euler palauttaa käännöksen, joka on x astetta x-akselilla, y astetta y-akselilla ja z astetta z-akselilla. Tässä tapauksessa se palauttaa verticalRotationin verran asteita x-akselille, koska x-akseli on horisontaalinen eli mitä käytetään ylös-alas katsomiseen.

```
verticalRotation -= Input.GetAxis("Mouse Y") * mouseSensitivity;
verticalRotation = Mathf.Clamp(verticalRotation, -upDownRange, upDownRange);
Camera.main.transform.localRotation = Quaternion.Euler(verticalRotation,0,0);
```

Seuraavaksi hahmon liikuttamisessa muuttujille forwardSpeed ja sideSpeed otetaan arvot pelaajan painaessa horisontaalisia tai vertikaalisia liikuntanäppäimiä, joiden arvot kerrotaan liikkumisnopeudella movementSpeed.

```
//Liikkuminen
float forwardSpeed = Input.GetAxis("Vertical")*movementSpeed;
float sideSpeed = Input.GetAxis("Horizontal")*movementSpeed;
```

VerticalVelocity lasketaan lisäämällä hahmon pudotessa painovoiman putoamiskiihtyvyys (9,81m/s^2)*kuluva aika deltaTime.

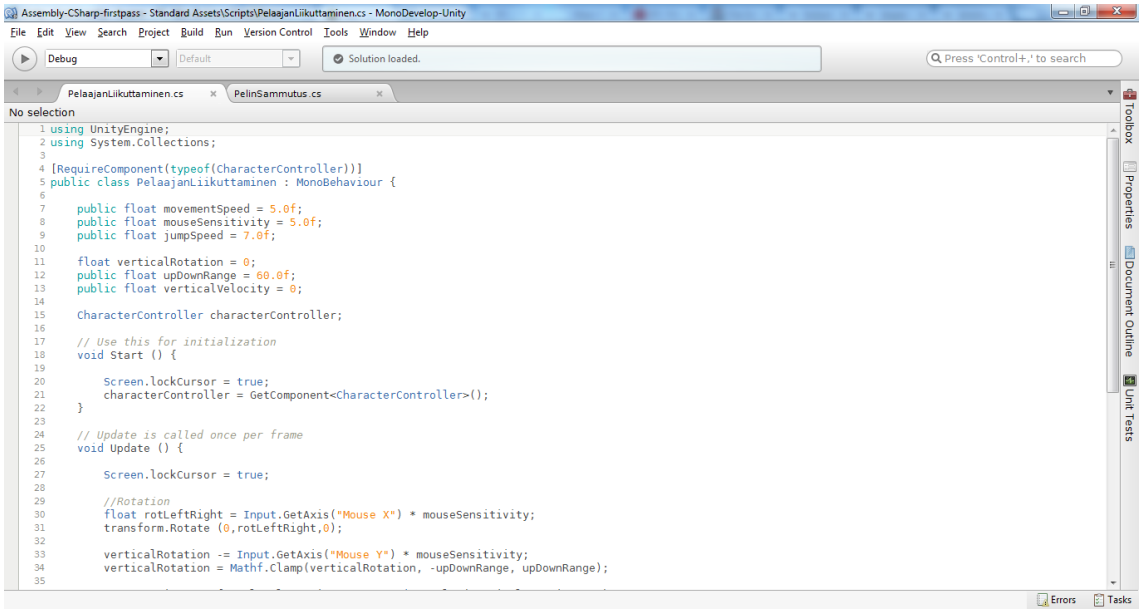
```
verticalVelocity += Physics.gravity.y * Time.deltaTime;
```

Lopuksi tehdään uusi vektori hahmon nopeudelle, jossa x-tason (horisontaalinen) liikkeen nopeutena toimii muuttuja sideSpeed, y-tason (ylös ja alas) liikkeen nopeutena toimii verticalVelocity ja z-tason (syvyyssuunnassa, tässä tapauksessa siis eteen- ja taaksepäin) liikkeen nopeutena toimii forwardSpeed. Nopeuden vektori speedin uudeksi arvoksi tulee speed kerrottuna kääntymisen muutoksella, transform.rotation. Lopuksi hahmon liikuttamiseen käytetyn characterControllerin liikkeeksi tulee nopeus speed kerrottuna kuluvalla ajalla Time.deltaTime.

```
Vector3 speed = new Vector3(sideSpeed, verticalVelocity, forwardSpeed);
speed = transform.rotation * speed;
characterController.Move(speed*Time.deltaTime);
```

Tämän päätteeksi hahmo on nyt valmis liikutettavaksi. Liikkumiseen käytettävä skripti täytyy vielä liittää pelihahmoon. Kuten kameran kanssa, skripti vain vedetään pelihahmon kohdalle ja se kiinnittyy pelihahmoon.

Koko liikkumiseen käytetty skripti on esitetty liite 2:n kohdassa 1. Lisäksi pieni ote liikkumiseen käytetystä skriptistä on kuvassa 25.



Kuva 25. Pelaajan liikuttamiseen käytetty skripti.

Kuvassa on samalla MonoDevelop 4.01, jota käytettiin pelin koodin kirjoittamiseen.

5.5 Ympäristön kanssa reagointi tietomallistudiossa

Timo Lehtoviidan toiveesta tietomallistudion peliympäristössä on pieniä interaktiivisia ominaisuuksia ja pelihahmolla itsellään on muutamia ominaisuksia, joita ei toisessa tietomallista tehdyssä pelissä ole. Huoneessa on tiettyjä objekteja, joita klikkaamalla voi lukea pienen kuvauksen kyseisestä laitteesta, minkä lisäksi huoneen valot voidaan sammuttaa ja laittaa taas päälle painamalla E-näppäintä valokatkaisimen vieressä. Lisäksi pelaaja voi zoomata pelin sisäistä näkymää. Näitä toimintoja on kuvattu seuraavissa alikappaleissa.

5.5.1 Objektien klikkaaminen

Huoneessa on 11 objektia joita pelaaja voi klikata:

- kannettava tietokone
- Smartboard -älynäyttö
- TV-monitori
- lähetin 1 (joka lähettää signaalia vastaanotin 1:een, joka on liitetty projektoriin)
- lähetin 2 (joka lähettää signaalia vastaanotin 2:een, joka on liitetty monitoriin)
- projektori
- vastaanotin 2, joka on liitetty monitoriin
- tulostin
- reititin
- lähetin 1:n kaukosäädin

- valokatkaisin

Siirtämällä pelissä olevan tähtäimen näiden objektien päälle ja pitämällä hiiren vasemmanpuoleisen painikkeen painettuna pohjaan peli avaa pienen laatikon, jossa lukee tietoa kyseisestä laitteesta.

Skripti aloitetaan määrittelemällä totuusarvo (”boolean”) nimeltään isClicked. Tämän arvoksi alustetaan epätosi, ”false”.

```
public bool isClicked = false;
```

Tämän jälkeen aina, kun hiiri painetaan pohjaan sen ollessa objektin kohdalla, isClickedin arvoksi asetetaan tosi. Kun hiiren painike ei enää ole pohjassa, isClickedin arvo palautuu epätodeksi.

```
public void OnMouseDown()
          {
                    isClicked = true;
                    }

          public void OnMouseUp()
          {
                    isClicked = false;
                    }
```

Lopuksi skripti rakentaa tiedon sisältävän laatikon. Jos isClicked on tosi, piirretään ruudulle pieni laatikko lähes ruudun keskelle. Tämä laatikko sisältää laitteen kuvauksen.

```
public void OnGUI()
          {
if (isClicked)
GUI.Label (new Rect ((Screen.width / 2 + 10 ), (Screen.height / 2 + 10), 400, 100), "Tämä on televisio. Sen vastaanotin on yhteydessä\n" + "lähetin #2:een. Tietokoneen kuvan saa näkymään monitorista.", "box");
                    }
```

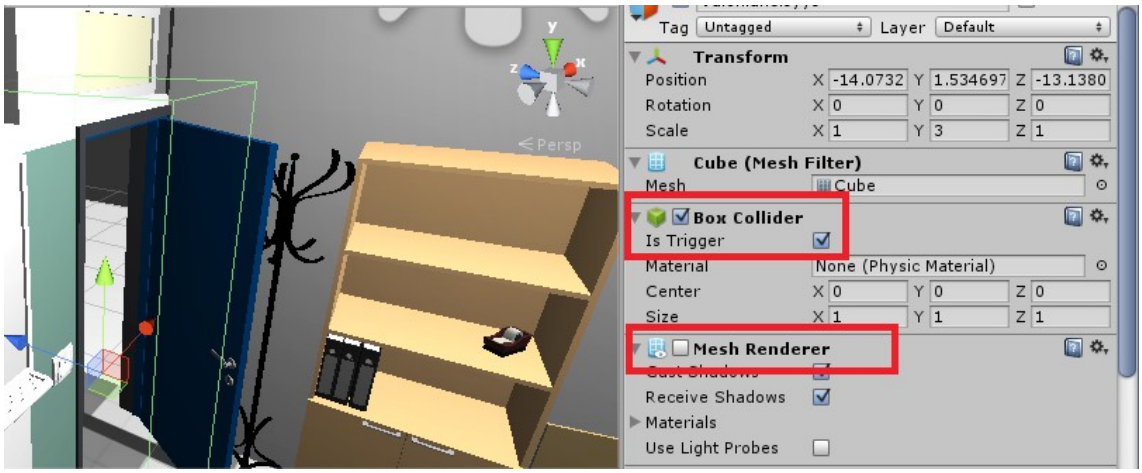
Tämä skripti toimii sellaisenaan kaikissa eri laitteissa. Skripti täytyy vain muistaa liittää laitetta vastaavaan peliobjektiin ja laitteen kuvaus täytyy vaihtaa laitekohtaiseksi.

Television klikkaamiseen tarkoitettu skripti on esitetty kokonaisuudessaan liite 2:n kohdassa 2.

5.5.2 Valojen laittaminen päälle ja pois

Huoneessa on neljä pistevaloa, jotka vastaavat huoneen kattolamppuja. Nämä valot voi sammuttaa ja laittaa takaisin päälle painamalla näppäimistön E-näppäintä valokatkaisimen vieressä. Jotta E-näppäintä ei voi painaa missä tahansa huonetta, täytyy valokatkaisimen viereen luoda erillinen liipaisin, ”Trigger”.

Tämä tehdään luomalla peliin uusi objekti, tässä tapauksessa kuutio, jonka kokoa kasvatetaan, kunnes se koetaan tarpeeksi suureksi jotta sitä voi käyttää liipaisimena. Kuutio täytyy myös asettaa niin sanotusti ”tyhjäksi”, jolloin se on täysin näkymätön pelissä eikä sitä pysty koskemaan pelissä (ettei hahmo tartu kiinni näkymättömään objektiin). Tämä tehdään valitsemalla liipaisimena toimivan kuution ”Mesh Renderer”, joka renderöi objektin ulkoasun pelissä, pois päältä. Lisäksi objekti täytyy määrittää liipaisimeksi. Tämä tehdään valitsemalla objektin ”Box Collider”-kohdasta ”Is Trigger” päälle. Nämä molemmat näytetään kuvassa 26.



Kuva 26. Box Collider päällä, Mesh Renderer pois päältä.

Kun liipaisin on näkymätön ja toimii liipaisimena, täytyy sille vielä kirjoittaa skripti, jolla valot sammutetaan. Tämä aloitetaan luomalla taulukko, joka pitää sisällään kaikki kentässä olevat valot. Tämän taulukon nimeksi annetaan ”lights”. Lisäksi koodissa täytyy ilmoittaa, mikä peliobjekti toimii liipaisimena. Tämä tehdään lisäämällä koodiin julkinen peliobjekti nimeltään ”Valonlaheisyys” (sama peliobjekti joka tätä ennen tehtiin).

```
public Light[] lights;
public GameObject Valonlaheisyys;
```

Tämän jälkeen pelaajan ollessa yhteydessä ”Valonlaheisyyden” kanssa (tarkemmin sanottuna pelaajan törmäilijän eli ”Colliderin” ollessa yhteydessa ”Valonlaheisyyden” liipaisimen kanssa) pelaajan painaessa E-näppäintä koodi suoritetaan. Tämä koodi vaihtaa kaikki ”lights”-taulukossa olevat valot päälle ja pois.

```
void OnTriggerStay(Collider Pelaaja)
          {
foreach (Light light in lights)
                    {
          if (Input.GetKeyDown(KeyCode.E))
                              {
                              light .enabled = !light .enabled;
                              }
                    }
          }
```

Tämän jälkeen koodi liitetään ”Valonlaheisyys”-peliobjektiin. ”Lights”-taulukolle täytyy antaa koko sen sammuttamien valojen määrälle (tässä opinnäytetyössä neljä). Tämän jälkeen jokainen huoneen valoista liitetään yhdeksi taulukon elementeistä. Lopuksi ”Valonlaheisyys”-objekti liitetään skriptin aktivoivaksi objektiksi.

Valojen sammuttamiseen tarkoitettu skripti on esitetty kokonaisuudessaan liite 2:n kohdassa 3.

5.5.3 Pelihahmon kyky zoomata

Pelihahmo voi zoomata näkymäänsä pelissä käyttämällä hiiren rullaa. Eteenpäin pyörittäminen lähentää näkymää ja taaksepäin pyörittäminen siirtää sitä kauemmas.

Tämä on yksi opinnäytetyön harvoista skripteistä, jotka tehtiin Javascript-ohjelmointikielellä. Aluksi skriptille määritetään kolme muuttujaa: minFov, maxFov ja sensitivity. Nämä säätelevät pelihahmon näkökenttää, ”Field of View” (eli pelaajan näkemän kuvan kulmaa, miten laajasti pelihahmo näkee). Sensitivity määrää asteikon, jonka mukaan zoomi kasvaa ja laskee.

```
var minFov: float = 15f;
var maxFov: float = 90f;
var sensitivity: float = 10f;
```

Tämän jälkeen luodaan uusi muuttuja fov, jonka arvoksi alustetaan pelin pääkameran näkökenttä asteina. Tämän jälkeen fov:in arvo lasketaan aina uudelleen kertomalla hiiren rullasta saatu arvo sensitivity-muuttujalla. Seuraavaksi fov:ia rajoitetaan; se ei voi mennä yli sille annetun maksimiarvon, eikä ali sille annetun minimiarvon. Lopuksi pelin pääkameran näkökentäksi asetetaan muuttuja fov, joka nyt lasketaan aina uudelleen joka ruutu per sekunti pelaajan käyttäessä hiiren rullaa.

```
function Update () {
  var fov: float = Camera.main.fieldOfView;
  fov -= Input.GetAxis("Mouse ScrollWheel") * sensitivity;
  fov = Mathf.Clamp(fov, minFov, maxFov);
  Camera.main.fieldOfView = fov;
}
```

Skripti liitetään pelin pääkameraan, jotta se saadaan toimintaan.

Zoomaukseen käytetty skripti esitetään kokonaisuudessaan liite 2:n kohdassa 4.

5.5.4 Pelin tähtäimen piirtäminen

Peliä pelatessaan pelaajan hiiri on näkymätön. Hiiri on keskitetty ruudun keskelle, mutta objektien aivan reunojen klikkaaminen voi olla hankalaa, jos hiiren sijainti joudutaan arvioimaan. Tätä varten peliin lisättiin tähtäin, joka pysyy koko ajan ruudun keskustassa.

Aluksi tähtäin piirrettiin Microsoftin Paint -kuvankäsittelyohjelmalla. Kun tämä oli tehty, tuotiin tähtäin uudeksi Assetiksi peliin ”Import New Asset”-toiminnolla. Tekstuurin piirtämiselle täytyi myös kirjoittaa skripti. Tässä skriptissä aluksi luodaan kaksi muuttujaa: ”crossHairTexture”, joka on kaksiulotteinen tekstuuri, ja ”position”, joka on laatikko. Näistä crossHairTexture tulee määrittämään pelissä käytettävän tähtäimen tekstuurin, joka tulee olemaan aikaisemmin piirretty tähtäin. Position taas tulee määrittämään tähtäimen sijainnin ja itse

tähtäimen sisältävän laatikon. Lisäksi alustetaan totuusarvo OriginalOn. Sen arvoksi annetaan tosi.

```
var crosshairTexture : Texture2D;

var position : Rect;

static var OriginalOn = true;
```

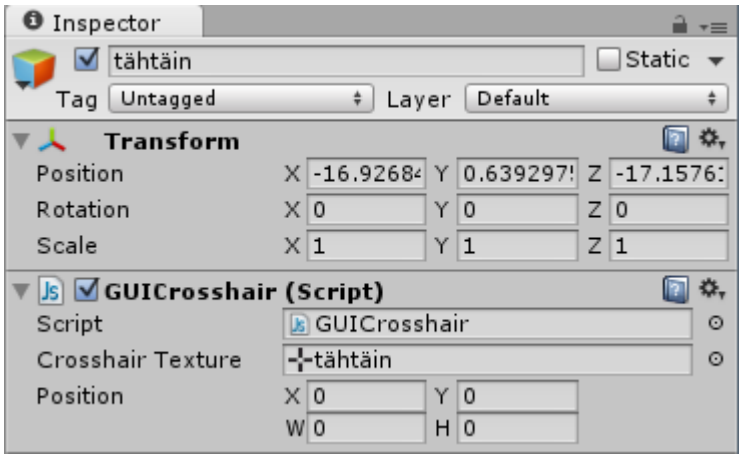
Koodin aluksi positionin sijainniksi lasketaan ruudun keskusta, josta vähennetään tähtäin-tekstuurin korkeus ja leveys. Tämä lasketaan ottamalla ruudun leveys ja korkeus, vähentämällä niistä tekstuurin leveys ja korkeus, ja jakamalla saatu sijainti kahdella. Lopuksi piirretään laatikko, jonka leveys ja korkeus ovat tähtäin-tekstuurin leveys ja korkeus.

```
function Start()
{
position = Rect((Screen.width - crosshairTexture.width) / 2,
(Screen.height - crosshairTexture.height) /2,
crosshairTexture.width, crosshairTexture.height);
}
```

Seuraavaksi tähtäin piirretään. Jos totuusarvo OriginalOn on tosi, piirretään pelinäkymään tekstuuri positionin sijaintiin ja sen varaamaan tilaan ja tekstuuriksi annetaan crossHairTexturessa annettu tekstuuri.

```
function OnGUI()
{
    if(OriginalOn == true)
    {
        GUI.DrawTexture(position, crosshairTexture);
    }
}
```

Lopuksi tähtäimen piirtämiseen luotu skripti liitetään uuteen peliobjektiin nimeltään ”tähtäin”. Aikaisemmin piirretty tähtäin-tekstuuri liitetään skriptissä pyydettyyn crossHairTexture-muuttujaan. Tämä esitetään kuvassa 27.



Kuva 27. Tähtäin-objektin tutkija-näkymä.

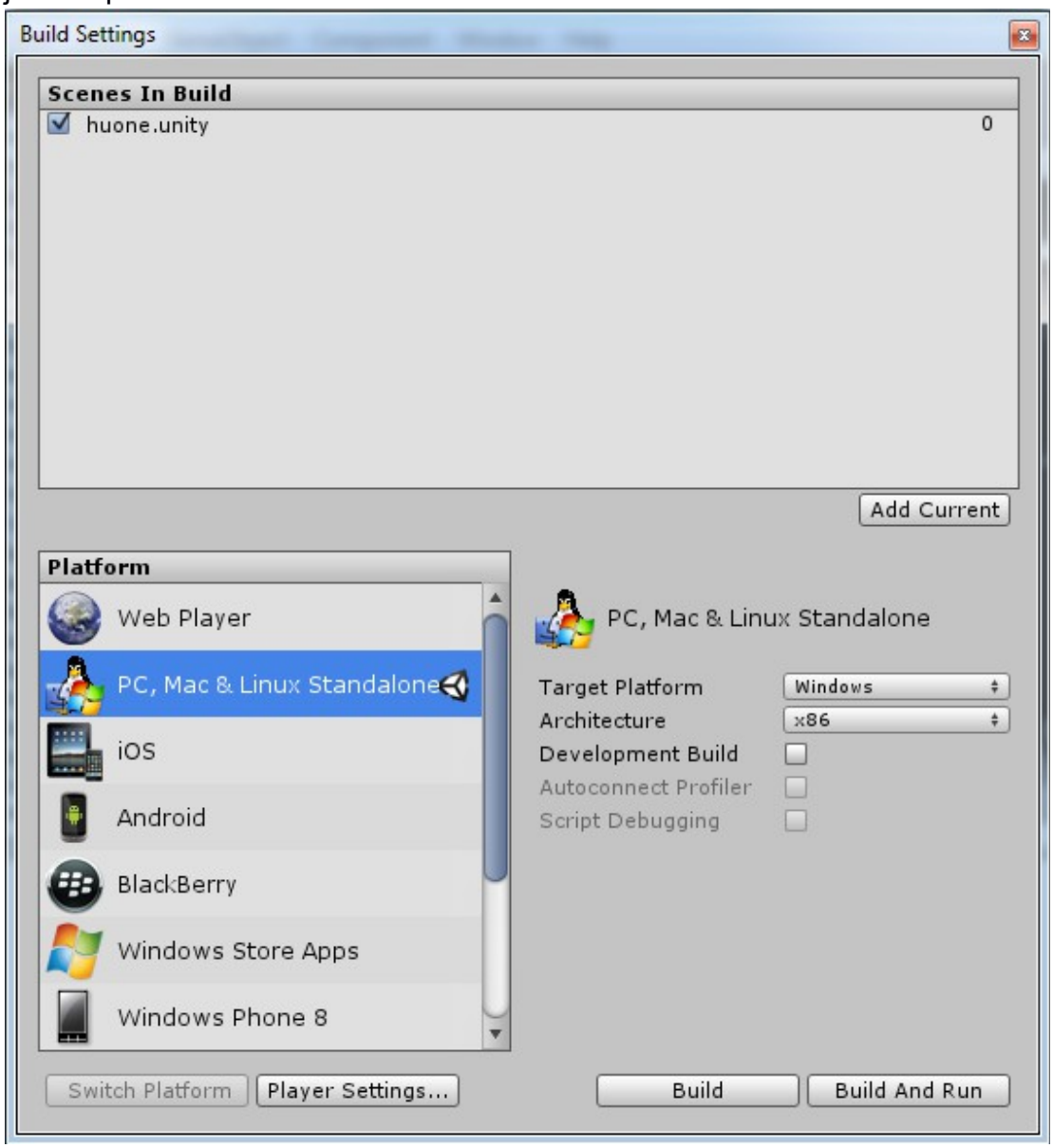
Piirrettävän tähtäimen sijaintia voisi vielä muuttaa vaihtamalla Position-tekstikenttien arvoja, mutta koska itse hiiri sijaitsee keskellä näyttöä, tämä ei kannata. Tähtäimen piirtämiseen käytetty skripti esitetään liite 2:n kohdassa 5.

5.6 Pelien julkaiseminen

Kun peliympäristö on saatu Unityyn, tehty siitä tutkittava, asetettu pelaajan vapaasti ohjattavissa oleva pelihahmo ympäristöön, ohjelmoitu pelaajan toiminnot sekä ympäristön interaktiivisuutta parantavat ominaisuudet, ovat pelit valmiita ”julkaistavaksi”. Unity antaa mahdollisuudet julkaista kehitetty peli monille eri alustoille, kuten Windows, älypuhelimet ja pelikonsolit. Tätä opinnäytetyötä varten kokeiltiin vain pelin Windows-rakennusta.

Unityn valikoista valitaan **File → Build Settings**. Tämä avaa Unityn rakennusvalikon, josta voidaan valita peliin tulevat kentät (Scenes). Seuraavaksi valitaan, mille alustalle pelin haluaa julkaista (tässä opinnäytetyössä Windows) ja valitun alustan asetukset. Lopuksi valitaan, halutaanko peli vain tehdä vai halutaanko se samalla suorittaa heti (Build ja Build And Run). Lisäksi alalaidassa on vielä Player Settings -valikko, jossa pääsee muun muassa valitsemaan pelin pikakuvakkeen ulkoasun, mitä asetuksia pelaaja voi muuttaa ja niin edelleen. Kuvassa 28 on kuvattu tämä julkaisuvalikko. Ylhäällä valitaan peliin sisällytettävät kentät, alavasemmalla

valitaan julkaisualusta ja alaoikealla ovat niin alustakohtaiset asetukset kuin julkaisupainikkeet.



Kuva 28. Pelin julkaisuvalikko

Kun nämä valinnat on tehty, peli rakennetaan. Opinnäytetyössä testatussa Windows-versiossa Unity luo kaksi eri kohdetta: pikakuvakkeen ja materiaalikansion. Peli käynnistetään pikakuvakkeesta, ja materiaalikansio sisältää kakki pelin tarvitsemat materiaalit (tekstuurit, mallit, skriptit...). Peli vaatii materiaalikansion samaan kansioon pikakuvakkeen kanssa toimiakseen. Muuten peli ei vaadi mitään eri ohjelmia toimiakseen. Tämä korostaa pelin helppokäyttöisyyttä tietomallin esittämisessä.

6 Yhteenveto ja pohdinta

Opinnäytetyössä testattiin ja opiskeltiin Unity-pelimoottorin ja 3ds Max Design 2012- sekä Blender -3D-mallinnusohjelmien perusteita. Lisäksi tutustuttiin tietomallien käsitteeseen ja niiden mahdollisiin käyttömahdollisuuksiin pelien kehityksessä.

Opinnäytetyön lopputuloksena on tämä raportti sekä kaksi Unitylla kehitettyä yksinkertaista peliä, joissa käyttäjä voi vapaasti tutkia kahdesta eri tietomallista rakennettuja peliympäristöjä. Raportti on tarkoitettu sekä pelialasta kiinnostuneille että tietomallien kehittäjille tai suunnittelijoille, jotka haluavat lisää tietoa kehittelemiensä mallien käyttömahdollisuuksista.

Opinnäytetyössä tehtyjä pelejä ja etenkin tietomallistudion peliä voidaan vielä kehittää huomattavasti. Tämä voi johtaa tietomallistudion jatkokehitykseen, jossa keskitytään enemmän rakennuksen tietomallien osaan huoneen hyödyntämisessä Unityssa. Yleisesti tässä vaiheessa keskityttäisiin enemmän tietomallien havainnollistamiseen Unityn avulla. Tietomallistudioon voidaan myös kehittää lisää ominaisuuksia ja toimintoja. Myös ympäristön lisäämistä tietomalleista saatuihin peliympäristöihin voidaan tarkastella.

Kuvat

Lähteet

Autodesk 2014a. History
http://www.autodesk.co.uk/adsk/servlet/index?siteID=452932&id=16032914. Luettu 28.04.2014.

Autodesk 2014b. 3ds Max Design vai 3ds Max
http://www.autodesk.fi/products/autodesk-3ds-max-design/compare/compare-products. Luettu 28.04.2014.

Blender 2013. History
http://www.blender.org/foundation/history/. Luettu 30.04.2014.

Brodkin, Jon 2013. How Unity3D became a Game-Development Beast
http://news.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. Luettu 30.04.2014.

BuildingSMART 2013. Industry Foundation Classes (IFC) data model
http://www.buildingsmart.org/standards/ifc. Luettu 1.05.2014.

IfcOpenShell 2014
http://ifcopenshell.org/. Luettu 01.05.2014.

M.A.D – Micro Aided Design – ArchiCAD
http://www.mad.fi/mad/archicad.html. Luettu 28.04.2014.

OpenBIM 2012. The open Toolbox for BIM – xBIM xPlorer
http://www.openbim.org/case-studies/ifc-viewer. Luettu 30.04.2014.

Unity Technologies 2013. License Comparisons
http://unity3d.com/unity/licenses. Luettu 28.04.2014.

Savisalo, Anssi 2012. Tietomallinnus – Kohti virtualisoitua suunnitteluympäristöä
http://www.kuntamarkkinat.fi/portals/2/Savisalo%20Anssi_Tietomallinnus_kuma_1309.pdf. Luettu 02.05.2014.

Ward, Jeff 2008. What is a Game Engine?
http://www.gamecareerguide.com/features/529/what_is_a_game.php?page=2. Luettu 01.05.2014.

Liite 1 Uuden tietomallin käyttöönottoon tarvittavat vaiheet

1. Uusi 3D-malli uudesta tietomallista.
2. Uusi 3D-malli Unityyn (viedään 3D-mallinnusohjelmasta sopivassa tiedostomuodossa → Export, tuodaan Unityyn → Import New Asset).
3. Uudesta 3D-mallista peliympäristö, peliympäristölle lisätään törmäyttimet ja rajat.
4. Peliympäristöön lisätään uudet valot tai vanhat valot siirretään uusille paikoilleen.
5. Tekstuurit kiinnitetään niitä vastaaviin objekteihin tai pintoihin.
6. Ympäristöön luodaan uusi pelaaja tai vanha pelaaja siirretään uudelle paikalleen (tällöin pelaajaan liittyvät komponentit ja skriptit pysyvät pelaajassa, muuten ne täytyy kiinnittää uudelleen).
7. Vanhat skriptit kiinnitetään uusiin skriptejä vastaaviin peliobjekteihin, joko skriptit tai peliobjektit nimetään oikein, jotta ne vastaavat toisiaan.

Liite 2 Pelissä käytettyjä skriptejä

1 Pelihahmon liikuttaminen

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(CharacterController))]
public class PelaajanLiikuttaminen : MonoBehaviour {

		public float movementSpeed = 5.0f;
		public float mouseSensitivity = 5.0f;
		public float jumpSpeed = 7.0f;

		float verticalRotation = 0;
		public float upDownRange = 60.0f;
		public float verticalVelocity = 0;

		CharacterController characterController;

		// Use this for initialization
		void Start () {

				Screen.lockCursor = true;
				characterController = GetComponent<CharacterController>();
		}

		// Update is called once per frame
		void Update () {

				Screen.lockCursor = true;

				//Rotation
				float rotLeftRight = Input.GetAxis("Mouse X") * mouseSensitivity;
				transform.Rotate (0,rotLeftRight,0);


				verticalRotation -= Input.GetAxis("Mouse Y") * mouseSensitivity;
				verticalRotation = Mathf.Clamp(verticalRotation, -upDownRange,
upDownRange);

				Camera.main.transform.localRotation =
Quaternion.Euler(verticalRotation,0,0);

				//Movement

				float forwardSpeed = Input.GetAxis("Vertical")*movementSpeed;
				float sideSpeed = Input.GetAxis("Horizontal")*movementSpeed;

				verticalVelocity += Physics.gravity.y * Time.deltaTime;

				if (characterController.isGrounded) {
```

```
                                        verticalVelocity = 0;
                                                }

                        if(characterController.isGrounded && Input.GetButton("Jump") )
                        {
                                        verticalVelocity = jumpSpeed;
                        }

                        Vector3 speed = new Vector3(sideSpeed, verticalVelocity,
forwardSpeed);
                        speed = transform.rotation * speed;

                        characterController.Move(speed*Time.deltaTime);
            }
}
```

2 TV:n klikkaaminen

```
using UnityEngine;

using System.Collections;

public class KlikkaaTV : MonoBehaviour {

            public bool isClicked = false;
            public string name = "Some name";
            // Use this for initialization
            void Start () {
//          if (gameObject.GetComponent (Collider) == null)
//
            gameObject.AddComponent (typeof(BoxCollider));
            }

            // Update is called once per frame
            void Update () {

            }
            public void OnMouseDown()
            {
                        isClicked = true;
                        }

            public void OnMouseUp()
            {
                        isClicked = false;
                        }

            public void OnGUI()
            {
                        if (isClicked)

                                        GUI.Label (new Rect ((Screen.width / 2 + 10 ),
(Screen.height / 2 + 10), 400, 100), "Tämä on televisio. Sen vastaanotin on yhteydessä\n" +
                                                    "lähetin #2:een. Tietokoneen kuvan saa
näkymään monitorista.", "box");
```

```
                        }
}
```

3 Valojen sammuttaminen

```
using UnityEngine;

using System.Collections;

public class LightSwitch : MonoBehaviour {

            public Light[] lights;
            public GameObject Valonlaheisyys;

//
//
//
//
            void Update ()
//
            {
                                                                                    }

            void OnTriggerStay(Collider Pelaaja)
            {

                        foreach (Light light in lights)
////
                        {
////
                                    if (Input.GetKeyDown(KeyCode.E))
////
                                    {
////
                                                light .enabled = !light .enabled;
////
                                    }
////
                        }
//
            }
}
```

4 Pelihahmon kyky zoomata

```
#pragma strict


var minFov: float = 15f;
var maxFov: float = 90f;
var sensitivity: float = 10f;

function Update () {
  var fov: float = Camera.main.fieldOfView;
```

```
  fov -= Input.GetAxis("Mouse ScrollWheel") * sensitivity;
  fov = Mathf.Clamp(fov, minFov, maxFov);
  Camera.main.fieldOfView = fov;
}
```

5 Tähtäimen piirtäminen

```
var crosshairTexture : Texture2D;

var position : Rect;
static var OriginalOn = true;

function Start()
{
    position = Rect((Screen.width - crosshairTexture.width) / 2, (Screen.height -
        crosshairTexture.height) /2, crosshairTexture.width, crosshairTexture.height);
}

function OnGUI()
{
    if(OriginalOn == true)
    {
        GUI.DrawTexture(position, crosshairTexture);
    }
}
```